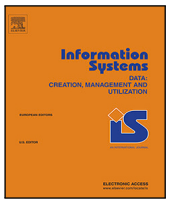
Information Systems 133 (2025) 102571

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is



Service composition for LTL_f task specifications

Giuseppe De Giacomo [a,b], Marco Favorito [c], Luciana Silo [b,d,*]

[a] University of Oxford, UK
[b] Sapienza University, Rome, Italy
[c] Banca d'Italia, Italy
[d] Camera dei deputati, Italy

ARTICLE INFO

Keywords:
Service composition
Linear Temporal Logic on finite traces
LTL_f synthesis

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 goals specified allowed traces declaratively via Linear Temporal Logic on finite traces (LTL_f). Specifically, we synthesize a controller to orchestrate the available services to produce a trace satisfying a specification in LTL_f. We demonstrate that this framework has several interesting applications, like Smart Manufacturing and Digital Twins.

1. Introduction

Service composition, which refers to the ability to generate new, more useful services from existing ones, addresses two key challenges. First, the synthesis, in order to synthesize, either manually or automatically, a specification of how to coordinate the component services to fulfill the client request. Secondly, the orchestration, i.e., how executing the previous obtained specification by suitably supervising and monitoring both the control flow and the data flow among the involved services [1].

Particularly interesting in this context is the so-called Roman Model [2–5] where services are *conversational*, i.e., have an internal state and are modeled as finite state machines (FSM), 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 composition aims at obtaining, given a (virtual) target service specifying a desired interaction with the client, a scheduler, called *orchestrator* that will use actions provided by existing services to implement action requests.

The original paper on the Roman Model [2] was awarded as the most influential paper of the decade at ICSOC 2013 and is, to date, the most cited paper in the ICSOC conference series. Moreover, it has been the inspiration for a line of work in AI on behavior composition where nondeterminism, in the sense of partial controllability as in Fully-Observable NonDeterministic (FOND) strong planning [6], had a prominent role [7].

Following recent advances in automatic orchestration and composition of software artifacts, a renewed interest in service composition *à la* Roman Model is recently stemming out of applications in smart manufacturing. Here, through digital twins technology, manufacturing devices can export their behavior 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., [8–10].

Interestingly, in the late 2000's the BPM community came up with a brilliant idea: give the rules that a process should satisfy and extract the process from the rules only [11,12]. In other words, the representation of the process is dropped explicitly, and LTL formulas are used to specify the allowed process traces. These applications are, therefore, 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 [13–15].

Hence, the target would ideally be specified in DECLARE, i.e., in Linear Temporal Logic on finite traces (LTL_f) [16], 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 [17,18]), and that only one action can be selected at each point in time [13]. This gives us a rich way to specify dynamic tasks that extend over time as testified by the DECLARE paradigm [13] itself, and the recent success of such kind of specification in an advanced form of BPM [15].

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

* Corresponding author at: Sapienza University, Rome, Italy.
E-mail address: silo@diag.uniroma1.it (L. Silo).

https://doi.org/10.1016/j.is.2025.102571
Received 1 October 2024; Received in revised form 24 April 2025; Accepted 2 May 2025
Available online 17 May 2025

formula, which in turn corresponds to implicitly specifying the transition system consisting of deterministic automaton corresponding to the LTL_f formula [16]. Then, after a preprocessing of the LTL_f specification to obtain the target transition system (i.e., the DFA corresponding to the formula) as in [16], the composition can essentially 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, *proactively* 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 two scenarios. In the "deterministic" case, the behavior of the available services is entirely predictable: given the current state of a service and the next action to be executed, the resulting successor state is uniquely determined. In contrast, the "nondeterministic" case introduces uncertainty: when given a current state and an action, a service may transition to one of several possible successor states. The latter kind of behavior is often referred to as *partially controllable* in the AI literature (akin to FOND strong planning), as in [3,7].

In contrast with the Roman model-based process-oriented framework, in our goal-oriented framework, the process actions are not requested by a third party, e.g., the user/client submitting specific requests to the process handler, possibly in a nondeterministic way. Instead, the execution order of the business process actions is controlled by the orchestrator we aim to synthesize. Yet, the adoption of a logical formalism as LTL_f allows to declaratively and succinctly specify the desired sequences of actions (reachability properties) while imposing constraints on undesired sequences of actions (safety properties). Additionally, the framework is expressive enough to model the nondeterministic behavior of the services to consider uncontrollable events in the model.

We argue that this variant of the service composition problem, *à la* Roman model, is particularly relevant to the broader field of Business Process Management and Modeling. As a supporting example, it is interesting to observe that the goal-oriented framework we are proposing shares many similarities with the *Case Management Model and Notation* (CMMN) [19,20], a graphical representation language standardized by the Object Management Group.[1] CMMN is a declarative approach to modeling knowledge-intensive business processes based on the handling of *cases*, requiring the execution of various activities that may be performed in an unpredictable order in response to evolving situations. Hence, CMMN diverges from traditional procedural modeling languages like the *Business Process Model and Notation* (BPMN), which follows an imperative paradigm. The CMMN language also exhibits *goal-orientedness*, thanks to the concept of *milestone*, an element that represents any achievable target during the lifetime of a CMMN case. Interestingly, as an empirical evaluation in the context of Knowledge-Intensive process modeling reports [21], experienced business process designers found that CMMN greatly supported the characteristics of unpredictability and goal-orientedness of processes [22]. Another study highlighted the decreased complexity of a CMMN model with respect to a BPMN model in case the process exhibits unpredictability [23]. We speculate that our composition framework can be applied to CMMN thanks to the shared focus on the declarative nature of the goal-oriented specification of the process and on the ability to capture nondeterministic events hindering its successful execution.

To provide a solution technique for the deterministic variant of our composition problem, from the LTL_f specification we compute in linear time a *symbolic representation* of the corresponding *Nondeterministic Finite Automaton* (NFA). We do so by adopting the Alternating Automaton (AFA) associated with the formula as the symbolic representation of the NFA. Then, we adapt the interleaving procedure introduced in [24] to encode the symbolic NFA corresponding to LTL_f temporally extended goals into special planning actions domains that encode all the possible service actions at each computation point. We implemented our approach and evaluated different heuristics and Torres and Baier's [24] encodings to show the feasibility of our solution technique. Unfortunately, this solution technique does not work for nondeterministic services. In this case, we cannot avoid determinization by subset construction of the NFA for the formula, since the NFA-based construction does not allow, in general, to properly encode the orchestrator's choices at each point in time. We exhibit a counterexample by showing how the technique can fail in the nondeterministic setting. Therefore, the solution for the nondeterministic variant of the composition problem will require a determinization of the NFA into a DFA, and then encoding the problem into a nondeterministic (adversarial) planning domain that encodes all the possible service actions at each point of the computation. In other words, we reduce the composition problem to a nondeterministic planning problem for LTL_f goals [17], by using a compilation technique proposed by Camacho and McIlraith [25].

In Fig. 1, we give a high-level overview of how our solution techniques work, both for the deterministic case (Fig. 1(a)) and the nondeterministic case (Fig. 1(b)). In the case services are deterministic (Fig. 1(a)), from the LTL_f goal formula φ we compute the AFA $\mathcal{A}^a_\varphi$, and from the community of deterministic services $\mathcal{C}_d = \{S_1, \ldots, S_n\}$ we compute a deterministic planning domain $\mathcal{D}^d_C$ which simulates the services' behaviors and the interaction with the orchestrator. Then, we compute the cross-product between $\mathcal{D}^d_C$ and $\mathcal{A}^a_\varphi$, according to the Torres and Baier's technique [24], to get the planning problem Γ^d_C with the new planning domain $\mathcal{D}^d_C$ that, differently from $\mathcal{D}^d_C$, also includes the computation of the LTL_f formula via on-the-fly construction of the associated NFA $\mathcal{A}^{nd}_\varphi$. If a successful plan π exists, then by the correctness of our construction, starting from π, we can compute a (deterministic) orchestrator γ that realizes the input specification. In the case services are nondeterministic (Fig. 1(b)), from the nondeterministic services community $\mathcal{C}_{nd}$ we compute the FOND domain $\mathcal{D}^{nd}_C$, which differently from $\mathcal{D}^d_C$ now it considers the nondeterministic behavior of services (when chosen by the orchestrator). Moreover, in this case, from the LTL_f goal formula φ we compute the DFA $\mathcal{A}^d_\varphi$. Finally, we take the *cross-product*[2] between the FOND planning domain $\mathcal{D}^{nd}_C$ and the DFA $\mathcal{A}^d_\varphi$ to get nondeterministic planning problem $\Gamma^{nd}_{\varphi,C}$ with the new nondeterministic planning domain $\mathcal{D}^{nd}_{\varphi,C}$ which also considers the evaluation of the goal formula φ. If a successful, strong plan[3] π exists, then by the correctness of our construction, starting from π, we can compute a (deterministic) orchestrator γ that realizes the input specification. The terms s_0 and G_φ denote the initial state and the set of goal states of the planning problem, respectively.

To our knowledge, this is the first work that formalizes and solves a goal-oriented variant of service composition à la Roman Model, where the target specification is given declaratively LTL_f. Our contributions are both theoretical and practical:

[1] https://www.omg.org/

[2] In the field of planning for temporally extended goals in LTL_f (or similar logical formalisms), with the term *cross-product* we refer to a technique that reduces the problem to a standard reachability planning task, but on a domain that synchronously evaluates both the original planning domain and the satisfaction of the goal formula. See [17] for an example of this idea in the context of FOND planning for LTL_f goals.

[3] In nondeterministic planning, *strong plans* are plans that are guaranteed to achieve the goal, see [26].

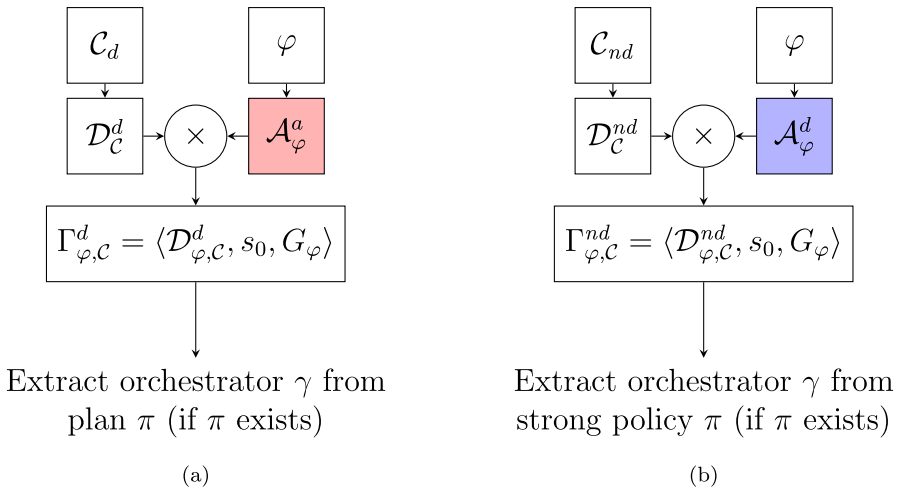


Fig. 1. A high-level picture of our solution techniques for the composition problem in the deterministic case (a) and the nondeterministic case (b). In Figure (a), from the deterministic service community $\mathcal{C}_d$, we compute the deterministic planning domain $\mathcal{D}^d_{\mathcal{C}}$, while from the goal LTL$_f$ formula φ we compute the alternating finite automaton $\mathcal{A}^a_{\varphi}$ (in red). Then, we compute the deterministic planning problem $\Gamma^d_{\varphi,\mathcal{C}}$ defined over the new deterministic planning domain $\mathcal{D}^d_{\varphi,\mathcal{C}}$ such that optimal solution plans correspond to successful orchestrators. In Figure (b) the procedure is similar, except that we start from a nondeterministic service community $\mathcal{C}_{nd}$, and instead of working with $\mathcal{A}^a_{\varphi}$ we compute the deterministic finite automaton $\mathcal{A}^d_{\varphi}$ (in blue); then, we reduce the composition problem to a FOND planning problem. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

- We introduce a new goal-oriented service composition framework based on the Roman model, where the orchestrator proactively controls the execution of services to fulfill an LTL$_f$-specified objective, rather than reacting to external requests. This departs from classical process-oriented models by embracing a declarative, proactive perspective.
- We analyze the goal-based LTL$_f$ composition problem in the special case when the services behave deterministically. We propose an automata-theoretic solution technique, where the composition problem is reduced to checking for non-emptiness of a particular DFA, of polynomial size of the goal formula, built from the input of the problem. We prove the correctness of the technique and analyze its computational cost, showing that we can solve it in EXPTIME;
- Then, we analyze the goal-based LTL$_f$ composition problem in its general form, i.e., when services can behave nondeterministically. To solve this variant of the problem, we rely on a reduction to a game-theoretic technique, where we have to find a winning strategy over a particular DFA game. We prove the correctness of the technique, showing that we can solve the general problem in 2EXPTIME. Moreover, we provide a counterexample that shows why we cannot use the technique for the deterministic variant;
- For both variants of the composition problem, we describe a reduction of the composition problem to an automated planning problem, aiming at a more practical and efficient approach than the theoretical construction since we can exploit performant planning systems;
- Finally, we describe our prototype implementation of the planning-based technique. Given the goal specification and the definition of the community of services, the tool generates a PDDL domain file and a PDDL problem file, which can be fed to a planning system. We empirically evaluate the compilation in a pair of industrial service composition scenarios taken from the literature;

Although this paper has a foundational nature, it formalizes the basic underlying mathematical framework and develops 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 [8,9,27].

The rest of the paper is organized as follows.

- In Section 2, we discuss related work in service composition, declarative process modeling, and automated planning.
- Section 3 introduces the necessary background on automata theory, LTL$_f$, FOND planning, and LTL$_f$ synthesis.
- In Section 4, we formalize our goal-oriented composition framework.
- Section 5 and Section 6 present our solution techniques for the deterministic and nondeterministic settings, respectively, including correctness proofs and complexity analyses.
- Section 7 describes our compilation into automated planning, while Section 8 details our implementation and empirical evaluation on realistic case studies.
- Finally, we conclude in Section 9 with a discussion and directions for future research.

2. Related work

In [28], 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 by Alves et al. [29] allows the modeling of nondeterministic behaviors but not of stateful services nor high-level temporal goal specification. Authors in [30] describe services as atomic actions where only I/O behavior is modeled, and the ontology is constituted by propositions and actions; hence, services are not stateful as ours. In [31], it is studied the problem of realizability of *agent planning programs*, high-level representations (in the form of transition systems) of the behavior of agents acting in a domain. States of the program represent decision points, and transitions are labeled by triples consisting of a guard, a maintenance goal and an achievement goal, over the domain. Each transition, in order to be executed, requires finding a plan in the underlying domain. Their mathematical framework is somewhat different from ours since agent programs are a form of procedural specifications, and the concept of service community is not explicitly considered, although their behavior could be modeled in the planning domain. Interestingly, the composition problem of deterministic agent behaviors [7] can be reduced to the realizability of agent planning programs, but in that framework, the target specification is process-oriented. In [32], a generalized two-player game framework for μ-calculus goal formulas is studied. Such framework captures the *multitarget composition* problem, which is close to our setting. However, a crucial difference lies in

the different logical formalisms adopted. In general, linear-time logics, and therefore LTL$_f$, can express most of the properties of interests, and it is more intuitive than branching logics [33]. Moreover, working directly with μ-calculus specification requires handling nested fixpoints and parity games, which are computationally expensive. De Giacomo et al. [34] 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 consider services which behave *adversarially*.

3. Preliminaries

3.1. Automata theory

A *deterministic finite automaton* (DFA) is a tuple $\mathcal{A}_\varphi = \langle \mathcal{P}, Q, q_0, F, \delta \rangle$ 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. Let the *extended transition function* $\delta^*(q, \epsilon) = q$ and $\delta^*(q, wa) = \delta(\delta^*(q, w), a)$. A *nondeterministic finite automaton* (NFA) is defined similarly to DFA except that δ is defined as a relation, i.e. $\delta \subseteq Q \times \mathcal{P} \times Q$. A word $w \in \mathcal{P}^*$ is *accepted* by a NFA or DFA $\mathcal{A}$ iff a sequence of states $q_0, \dots, q_l$ exists in Q such that: (*i*) $q_{i+1} \in \delta(q_i, a_i)$ for $i = 0, \dots, l-1$; and (*ii*) $q_l \in F$. With $\mathcal{L}(\mathcal{A})$, we denote the set of words accepted by $\mathcal{A}$.

3.2. Linear temporal logic on finite traces

LTL$_f$ is a variant of Linear Temporal Logic (LTL) interpreted over finite traces, instead of infinite ones [35]. 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 $\Diamond\varphi \equiv \mathit{true}\,\mathcal{U}\,\varphi$; 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 $\varphi_1 \,\mathcal{W}\, \varphi_2 \equiv (\varphi_1 \,\mathcal{U}\, \varphi_2 \vee \Box\varphi_1)$ (φ_1 holds until φ_2 or forever).

LTL$_f$ formulas are interpreted on finite traces. A finite trace over propositions $\mathcal{P}$ is a sequence $a = a_0 \dots a_{l-1}$ where $a_i \subseteq \mathcal{P}$, and l is the length of the trace. The empty trace is denoted with ϵ. Given a finite (possibly empty) trace a, we define when an LTL$_f$ formula φ is true in a at the instant $i \in \{0, \dots, l-1\}$, denoted by $a, i \models \varphi$, by inductively considering subformulas as follow [36,37]:

- $a, i \models tt$;
- $a, i \models p$ iff $a_i \in p$;
- $a, i \models \neg\varphi$ iff $a, i \not\models \varphi$;
- $a, i \models \varphi_1 \wedge \varphi_2$ iff $a, i \models \varphi_1$ and $a, i \models \varphi_2$;
- $a, i \models \bigcirc\varphi$ iff $i < l-1$ and $a, i+1 \models \varphi$;
- $a, i \models \varphi_1 \,\mathcal{U}\, \varphi_2$ iff there exists j with $i \le j < l$ such that $a, j \models \varphi_2$, and for all k with $i \le k < j$ we have that $a, k \models \varphi_1$.

Whenever $a, 0 \models \varphi$ holds, we say that a is a *model* of φ, or that a *satisfies* φ. We also write $a \models \varphi$ as an abbreviation for $a, 0 \models \varphi$. For simplicity, here we consider the semantics on *simple* (possibly empty) finite traces, also known as *process traces* [37],[4] where exactly one proposition can be true at any instant of time, i.e. $|a_i| = 1$ for all $i = 0, \dots, l-1$.

An LTL$_f$ formula can be transformed into an equivalent NFA $\mathcal{A}_\varphi$ defined on the same alphabet of φ in at most EXPTIME and into an equivalent DFA in at most 2EXPTIME [35].

[4] While the choice between finite-trace and process-trace semantics does not affect the generality of our framework, we prefer process-trace semantics as it aligns more naturally with our context. In this case, the traces result from the services' actions, which are more effectively represented as atomic propositions rather than in a factorized form.

3.3. FOND planning

A *Fully-Observable Non-Deterministic* (FOND) domain model can be formalized as a tuple $\mathcal{D} = \langle \mathcal{F}, A, pre, eff \rangle$ 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 $\{eff_1, \dots, eff_n\}$ of effects, respectively. Each effect $eff_i \in eff(a)$ is a set of conditional effects each of the form $c \rhd e$, 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 $\emptyset \rhd e$. 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 $c \rhd e$ 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 $eff(a)$. Let $eff_i \in eff(a)$ 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 $c \rhd e \in eff_i$ triggered in s deletes it ($\neg f \in e$) or (*ii*) there is a conditional effect $c \rhd e \in eff_i$ triggered in s that adds it ($f \in e$). In case of conflicting effects, similarly to other works [38], we assume delete-before-adding semantics [39]. We use $\delta(s, a)$ to denote the set of possible successor states $\{s'_1, \dots, s'_n\}$ obtained by executing a in s. Note that if $s \not\models pre(a)$ then $\delta(s, a) = \emptyset$. 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 $s_0, a_0, s_1, a_1, \dots$ 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 $\pi(u) = \perp$. A trace τ is *generated* by π, or simply an π-*trace*, if (i) if $s_0, a_0, \dots, s_i, a_i$ is a prefix of τ then $\pi(s_0 s_1 \dots s_i) = a_i$, and (ii) if τ is finite, say $\tau = s_0, a_0, \dots, a_{n-1}, s_n$, then $\pi(s_0 s_1 \dots s_n) = \perp$. A strategy π is a *(strong) solution* to Γ if every π-trace is a finite trace τ such that $s_n \models G$.

3.4. LTL$_f$ synthesis

Let φ be an LTL$_f$ formula over $\mathcal{P} = \mathcal{X} \cup \mathcal{Y}$, and $\mathcal{X}$, $\mathcal{Y}$ are two disjoint sets of propositional variables controlled by the *environment* and the *agent*, respectively. The *synthesis* problem $(\varphi, \mathcal{X}, \mathcal{Y})$ consists in computing a strategy $g : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$, such that for an arbitrary infinite sequence $\lambda = X_0, X_1, \dots, \in (2^{\mathcal{X}})^\omega$, we can find $k \ge 0$ such that $\rho^k \models \varphi$, where $\rho^k = (X_0 \cup g(\epsilon)), (X_1 \cup g(X_0)), \dots, (X_k \cup g(X_0, X_1, \dots, X_{k-1}))$. If such a strategy does not exist, then φ is unrealizable. LTL$_f$ synthesis can be solved by reducing to an adversarial reachability game on the corresponding DFA [40]. Hence, a strategy can also be represented as a finite-state controller $g : S \mapsto 2^{\mathcal{Y}}$, where S denotes the state space of the DFA.

4. Service composition for LTL$_f$ tasks

We start by presenting our service composition framework. Unlike the classical Roman model, we do not have an explicit specification of the target service to realize, but rather a high-level specification of a task to accomplish expressed as an LTL$_f$ formula. We want to accomplish such a task despite the available services having non-deterministic behavior, as in [3].

4.1. The framework

Following the Roman Model [2,3,7], each *(available) service* is defined as a tuple $S = \langle \Sigma, A, \sigma_0, F, \delta \rangle$ 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. If for all $\sigma \in \Sigma$ and $a \in A$ we have $|\delta(\sigma, a)| = 1$, then we say that the service S is *deterministic*; otherwise we say that S is a *nondeterministic* service.

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 $C = \{S_1, \ldots, S_n\}$, where each set of actions $A_i \subseteq A$, a *trace of* 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 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, $\sigma_{i,k+1} \in \delta_i(\sigma_{ik}, a_k)$ 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. $\mathsf{states}(t) = (\sigma_{10} \ldots \sigma_{n0}), (\sigma_{11} \ldots \sigma_{n1}), \ldots$. 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 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 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 $\gamma((\sigma_{10} \ldots \sigma_{n0}) \ldots (\sigma_{1m} \ldots \sigma_{nm})) = \perp$. Note that if the services are nondeterministic, we can have many executions for the same orchestrator despite the orchestrator being a deterministic function, whereas for deterministic services only one execution is possible. With Out_γ, we denote the set of executions of γ with C starting from the initial state $(\sigma_{10}, \ldots, \sigma_{n0})$, i.e. an execution $t = (\sigma_{10}, \ldots, \sigma_{n0}), (a_1, o_1), \ldots, (a_m, o_m), (\sigma_{1m}, \ldots, \sigma_{nm})$ is in Out_γ iff for all k with $1 \leq k \leq m$, we have $\gamma((\sigma_{10}, \ldots, \sigma_{n0}), \ldots, (\sigma_{1,k-1}, \ldots, \sigma_{n,k-1})) = (a_k, o_k)$ and $\delta_{o_k}(a_k, \sigma_{o_k,k})$ is defined. If services are deterministic, we have $|\mathsf{Out}_\gamma| = 1$.

In the following, we will formally state our problem of interest in the *goal-based* variant, as opposed to the classical process-based variant.

4.2. Goal-based LTL$_f$ service composition

Intuitively, the goal-based variant of our service composition problem consists in finding an orchestrator such that, for all possible executions t of the service community C, we have that the sequence of actions $\mathsf{actions}(t)$ satisfies the LTL$_f$ task φ. Formally, 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* C if all its executions are finite traces t that are successful. Note that the orchestrator, at every step, chooses the action and the (index of the) service to which the action is delegated. In doing so, it guarantees the (complete) 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 and when the orchestrator stops, all services are left in their final states. Then, the composition problem is:

Problem 1 (*Composition for* LTL$_f$ *Goal Specifications*)**.** Given the pair (C, φ), where φ is an LTL$_f$ goal specification over the set of propositions A, and C is a community of n services $C = \{S_1, \ldots, S_n\}$, compute, if it exists, an orchestrator γ that realizes φ.

Example 1. We present a didactic example of using our framework, inspired by the "garden bots system" example [41]. 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 Fig. 2 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. An example of a successful history is:

$$(a_0, b_0, c_0) \xrightarrow{(clean,1)} (a_1, b_0, c_0) \xrightarrow{(water,2)} (a_1, b_0, c_0) \xrightarrow{(pluck,3)}$$
$$(a_1, b_0, c_1) \xrightarrow{(empty,1)} (a_0, b_0, c_1) \xrightarrow{(empty,3)} (a_0, b_0, c_0)$$

5. Solution for goal-based LTL$_f$ composition of deterministic services

In the deterministic case, to synthesize the orchestrator, we rely on a reduction to standard automata-theoretic techniques, i.e.: (*i*) we build a DFA $\mathcal{A}_{\varphi,C}$ on words of pairs $(a, i) \in A \times \{1, \ldots, n\}$, where (a, i) means "execute action a with the ith service"; (*ii*) find an accepting word $w \in \mathcal{L}(\mathcal{A}_{\varphi,C})$, which by construction is such that φ is satisfied and services end in an accepting state; (*iii*) from this accepting word, we build the actual orchestrator γ. 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 $\mathcal{A}_{\varphi,C}$; (4) the latter DFA can be seen as a proxy to find successful executions, namely a word w is accepted by $\mathcal{A}_{\varphi,C}$ iff the history corresponding to w is successful; (5) if such a word w is found, we can derive an orchestrator that realizes φ from w. 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 automata on finite words [16]. The alphabet of the NFA is A since we assume process trace semantics.

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 nondeterminism of $\mathcal{A}_\varphi$ is canceled 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 $a_0, \ldots, a_m$ accepted by the NFA $\mathcal{A}_\varphi$, i.e. satisfying the formula φ, there exists a corresponding alternating sequence $q_0, a_0, \ldots, q_{m+1}$ accepted by the DFA $\mathcal{A}_{\mathsf{act}}$, and vice versa. It can be shown that:

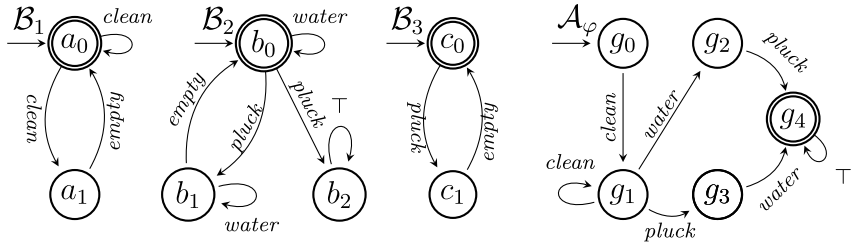
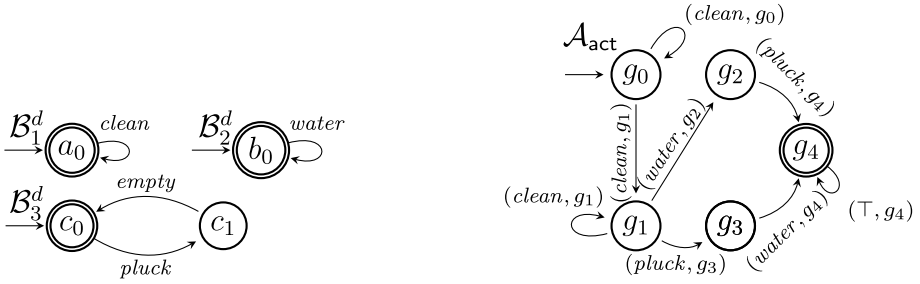


Fig. 2. From left to right: the three available bots of the garden bot systems, and the automaton $\mathcal{A}_\varphi$ of the LTL$_f$ goal.



(a) A simplified community of deterministic services for the Garden Bot scenario for Example 2.

(b) The controllable DFA corresponding to the DFA $\mathcal{A}_\varphi$ in Figure 2.

Fig. 3. A simplified community of deterministic services for the Garden Bot scenario for Example 2.

Proposition 1. *$a_0 \dots a_m \in \mathcal{L}(\mathcal{A}_\varphi)$ iff $(a_0, q_1) \dots (a_m, q_{m+1}) \in \mathcal{L}(\mathcal{A}_{\text{act}})$, for some $q_1 \dots q_{m+1}$.*

Proof. By definition, $a_0 \dots a_m \in \mathcal{L}(\mathcal{A}_\varphi)$ iff there exist a run $r = q_0, q_1 \dots q_{m+1}$ s.t. for $0 \le k \le m$, $\delta(q_k, a_k) = q_k$ and $q_{m+1} \in F$. Consider the word $w' = (a_0, q_1) \dots (a_m, q_{m+1})$. By construction of $\mathcal{A}_{\text{act}}$, w' induces a run $r^d = r$. Since $q_{m+1} \in F$ by assumption, r^d is an accepting run for $\mathcal{A}_{\text{act}}$, and therefore $w' \in \mathcal{L}(\mathcal{A}_{\text{act}})$ is accepted. The other direction follows by construction because, if $(a_0, q_1) \dots (a_m, q_{m+1}) \in \mathcal{L}(\mathcal{A}_{\text{act}})$, then by construction $q_0 \dots q_{m+1}$ is a run of $\mathcal{A}_\varphi$ over word $a_0 \dots a_m$, and since $q_{m+1} \in F$ by assumption, $a_0 \dots a_m \in \mathcal{L}(\mathcal{A}_\varphi)$. □

Example 2. Consider a simplified version of Example 1, where all the services are deterministic. The new community of deterministic services is shown in Fig. 3(a). As the NFA of the goal formula, we consider the DFA shown in Fig. 2 but with a self-loop on g_0 with label *clean* to add a simple form of nondeterminism, for didactic purposes. The corresponding controllable DFA $\mathcal{A}_{\text{act}}$ is shown in Fig. 3(b).

Step 3. Then, given $\mathcal{A}_{\text{act}}$ and $\mathcal{C}$, we build the *composition* DFA $\mathcal{A}_{\varphi,\mathcal{C}} = (A', Q', q_0', F', \delta')$ as follows:

- $A' = \{(a, q, i) \mid (a, q, i) \in A \times Q \times \{1, \dots, n\}\}$;
- $Q' = Q \times \Sigma_1 \times \cdots \Sigma_n$;
- $q_0' = (q_0, \sigma_{10} \dots \sigma_{n0})$;
- $F' = F \times F_1 \times \cdots \times F_n$;
- $\delta'((q, \sigma_1 \dots \sigma_i \dots \sigma_n), (a, q', i)) = (q', \sigma_1 \dots \sigma_i' \dots \sigma_n)$ iff $\delta_i(\sigma_i, a) = \sigma_i'$, and $\delta_{\text{act}}(q, (a, q')) = q'$.

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 $(a, q, i) \in A'$. Note that at any step, there is only one successor state for the chosen service i since δ_i is a deterministic function. 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 $(a_0, q_1, o_0) \dots (a_m, q_{m+1}, o_m) \in A'^*$, which induces the run $r = (q_0, \sigma_{10} \dots \sigma_{n0}), \dots, (q_{m+1}, \sigma_{1,m+1} \dots \sigma_{n,m+1})$ over $\mathcal{A}_{\varphi,\mathcal{C}}$, we define the history $h = \tau_{\varphi,\mathcal{C}}(w) = (\sigma_{10} \dots \sigma_{n0}), (a_0, o_0), \dots, (\sigma_{1,m+1} \dots \sigma_{n,m+1})$.

Example 3. Continuing with Example 2, now we construct the composition DFA starting from the deterministic services in Fig. 3(a) and the controllable DFA in Fig. 3(b). The result is shown in Fig. 4.

Step 4. We can exploit the DFA $\mathcal{A}_{\varphi,\mathcal{C}}$ to find a sequence of orchestrator actions such that the corresponding execution is successful. To do so, we can check for non-emptiness of $\mathcal{A}_{\varphi,\mathcal{C}}$. Intuitively, this works because $\mathcal{A}_{\varphi,\mathcal{C}}$ encodes the interaction between the orchestrator and the community of services. More precisely, it can be shown that there is a tight relationship between accepting runs of $\mathcal{A}_{\varphi,\mathcal{C}}$ and successful histories with $\mathcal{C}$ for specification φ:

Proposition 2. *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 $w \in \mathcal{L}(\mathcal{A}_{\varphi,\mathcal{C}})$.*

Proof. We prove both directions of the equivalence separately.

($\Leftarrow$) Let w be a word $w = (a_0, q_1, o_0) \dots (a_m, q_{m+1}, o_m) \in A'^*$. Assume $w \in \mathcal{L}(\mathcal{A}_{\varphi,\mathcal{C}})$. We have that the induced run $r = (q_0, \sigma_{10} \dots \sigma_{n0}), \dots, (q_{m+1}, \sigma_{1,m+1} \dots \sigma_{n,m+1})$ over $\mathcal{A}_{\varphi,\mathcal{C}}$ is accepting. This means that $q_{m+1} \in F$ and $\sigma_{i,m+1} \in F_i$ for all i. Now consider the history $h = \tau_{\varphi,\mathcal{C}}(w) = (\sigma_{10} \dots \sigma_{n0}), (a_0, o_0), \dots (\sigma_{1,m+1} \dots \sigma_{n,m+1})$. For every $0 \le k \le m$, we have by definition of δ' that $\delta_i(\sigma_{i,k}, a_k) = \sigma_{i,k+1}$ if $i = o_k$, and $\sigma_{i,k} = \sigma_{i,k+1}$ otherwise. Moreover, $o_k \in \{1 \dots n\}$, $a_k \in A$ and $\sigma_{ik} \in \Sigma_i$, for all $i \in \{1, \dots, n\}$ and $0 \le k \le m$. Therefore, h is a valid history. Moreover, h is a successful history because $q_{m+1} \in F$ iff $a_0, \dots, a_m \models \varphi$, and $\sigma_{i,m+1} \in F_i$ for all i.

($\Rightarrow$) Assume that $h = (\sigma_{10} \dots \sigma_{n0}), (a_0, o_0), \dots, (\sigma_{1,m+1} \dots \sigma_{n,m+1})$ is a successful history with $\mathcal{C}$ Then we both have that (*i*) $\text{actions}(h) = a_0, \dots, a_m \models \varphi$ and (*ii*) $\sigma_{i,m+1} \in F_i$ for all i. By construction of the NFA $\mathcal{A}_\varphi$, proposition (*i*) implies that there exists a run $q_0, \dots, q_{m+1}$ where $q_{m+1} \in F$. Now consider the sequence $w = (a_0, q_1, o_0), \dots, (a_m, q_{m+1}, o_m)$. By construction, we have $\tau_{\varphi,\mathcal{C}}(w) = h$. Moreover, from proposition (*i*) and proposition (*ii*), it follows that $w \in \mathcal{L}(\mathcal{A}_{\varphi,\mathcal{C}})$. This concludes the proof. □

Proposition 2 gives us a method to compute whether there exists an orchestrator that realizes φ with service community $\mathcal{C}$. If $\mathcal{A}_{\varphi,\mathcal{C}}$ is not empty, then let $w \in \mathcal{L}(\mathcal{A}_{\varphi,\mathcal{C}})$, and let $h = \tau_{\varphi,\mathcal{C}}(w)$ be its corresponding

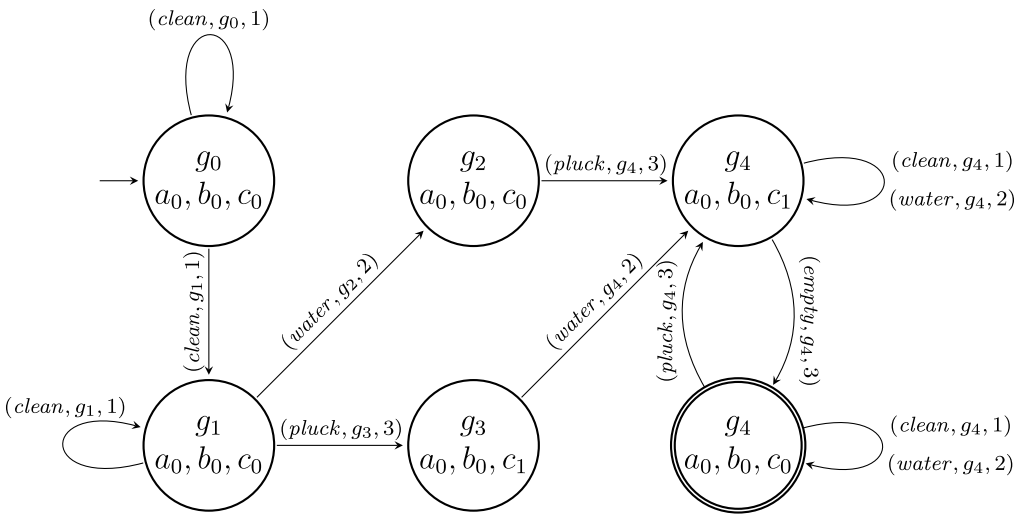


Fig. 4. The composition DFA for Example 3.

history. By Proposition 2, we have that h is also a successful history. Otherwise, if $\mathcal{L}(\mathcal{A}_{\varphi,\mathcal{C}}) = \emptyset$, then from Proposition 2 it follows that *no* history h is successful.

Step 5. Given an accepting word w, we can obtain an orchestrator that realizes the specification. Then, for every sequence w of the form $w = (a_0, q_1, o_0) \dots (a_k, q_{k+1}, o_k)$, where a_k, q_{k+1} and o_k are, respectively, the action, next NFA state, and service index chosen at the kth step ($k \geq 0$), we define the orchestrator $\gamma((\sigma_{10} \dots \sigma_{n0}), (\sigma_{11} \dots \sigma_{o_1,1} \dots \sigma_{n1}), \dots, (\sigma_{1k} \dots \sigma_{o_k,k} \dots \sigma_{n,k})) = (a_k, o_k)$, and whenever the trace satisfies the successful condition, we have that the next choice is $\perp$. Note that since the services are deterministic, the sequence of the services' states is fully determined by the initial services' states and the sequence of orchestrator actions (a_k, o_k) taken. Moreover, the $\mathcal{A}_{\mathsf{act}}$-state component of each step of w is projected out since its role in the solution was to find, in an automata-theoretic fashion, an action sequence $a_0, \dots, a_m$ that satisfies φ.

In the following, we show the correctness of our approach, i.e., that we can reduce the problem of service composition for LTL$_f$ goal specifications, in the deterministic case, to find an accepting trace of the DFA $\mathcal{A}_{\varphi,\mathcal{C}}$.

Theorem 1. *Let the composition problem be $\langle \varphi, \mathcal{C} \rangle$. Realizability can be solved by checking whether $\mathcal{L}(\mathcal{A}_{\varphi,\mathcal{C}}) \neq \emptyset$.*

Proof. We prove soundness and completeness separately.

Soundness. Assume there exists a word $w = (a_0, q_1, o_0) \dots (a_m, q_{m+1}, o_m)$ such that $w \in \mathcal{L}(\mathcal{A}_{\varphi,\mathcal{C}})$; we aim to show that there exists an orchestrator γ that realizes φ. By Proposition 2, and from the hypothesis, it follows that the history $h = \tau_{\varphi,\mathcal{C}}(w) = (\sigma_{10} \dots \sigma_{n0}), (a_0, o_0), \dots, (a_m, o_m), (\sigma_{1,m+1} \dots \sigma_{n,m+1})$ is successful, which in turn means that $a_0, \dots, a_m \vDash \varphi$ and that $\sigma_{i,m+1} \in F_i$ for $i = 1, \dots, n$. Now let γ be defined as in Step 5. Since γ is such that h is an execution of γ, we have that $h \in \mathsf{Out}_\gamma$. Moreover, since γ is a deterministic function, and the services' transition functions δ_i are deterministic, we also have that $|\mathsf{Out}_\gamma| = 1$. Therefore, since h is the unique execution of γ with community $\mathcal{C}$, and h is successful, we have that γ is a solution for the composition problem $\langle \varphi, \mathcal{C} \rangle$.

Completeness. Assume there exists an orchestrator γ that realizes φ with community $\mathcal{C}$; starting from γ, we construct a word w such that $w \in \mathcal{L}(\mathcal{A}_{\varphi,\mathcal{C}})$. Let $h = (\sigma_{10} \dots \sigma_{n0}), (a_0, o_0), \dots (a_m, o_m), (\sigma_{1,m+1} \dots \sigma_{n,m+1})$ be the execution of γ over service community $\mathcal{C}$. Since h is a successful execution, by definition it means that (*a*) $a_0, \dots, a_m \vDash \varphi$ and (*b*) $\sigma_{i,m+1} \in F_i$ for $i = 1, \dots, n$. From (*a*), it follows that $a_0, \dots, a_m \in \mathcal{L}(\mathcal{A}_\varphi)$, by the correctness of the construction of $\mathcal{A}_\varphi$. This means that there exists an accepting run $q_0, \dots, q_{m+1}$ corresponding to the execution of $a_0, \dots, a_m$ in $\mathcal{A}_\varphi$; in particular, we have that $q_{m+1} \in F$. Now, consider the word $w = (a_0, q_1, o_0), \dots, (a_m, q_{m+1}, o_m)$. By easy inspection, it holds that $h = \tau_{\varphi,\mathcal{C}}(w)$, since all terms a_k and o_k ($k \in [0, m]$) were taken precisely from h. Moreover, we claim that $w \in \mathcal{L}(\mathcal{A}_{\varphi,\mathcal{C}})$. This holds by construction of the transition function of $\mathcal{A}_{\varphi,\mathcal{C}}$, δ' and by definition of F'. In particular, the sequence of Q'-states $r = (q_0, \sigma_{10} \dots \sigma_{n0}), \dots, (q_{m+1}, \sigma_{1,m+1} \dots \sigma_{n,m+1})$ is a valid run with input w, since for all $k \in [0, m]$, we have that $\delta_{o_k}(\sigma_{o_k,k}, a_k) = \sigma_{o_k,k+1}$ by hypothesis that h is a valid execution, and $\delta_{\mathsf{act}}(q_k, (a_k, q_{k+1})) = q_{k+1}$ by definition of $\mathcal{A}_{\mathsf{act}}$ and from the above. Intuitively, r visits the states of $\mathcal{A}_{\varphi,\mathcal{C}}$ according to the services' transition functions δ_i and the transition function of the controllable DFA $\mathcal{A}_{\mathsf{act}}$; and therefore, it is a valid run of $\mathcal{A}_{\varphi,\mathcal{C}}$ To see that r is also an accepting run, i.e., $(q_{m+1}, \sigma_{1,m+1} \dots \sigma_{n,m+1}) \in F'$, it is enough to observe that $\sigma_{i,m+1} \in F_i$ for all $i = 1, \dots, n$ by (*b*), and $q_{m+1} \in F$ from above. This proves that $w \in \mathcal{L}(\mathcal{A}_{\varphi,\mathcal{C}}) \neq \emptyset$. □

Considering the cost of each step above, we get the following upper bound for the worst-case computational cost.

Theorem 2. *Service composition for LTL$_f$ goals 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.*

Proof. We take into the account the costs of each step of the solution technique explained in this section. The transformation of an LTL$_f$ formula φ into a NFA is exponential in the size of the formula since the resulting automaton must account for all possible temporal behaviors encoded within the LTL$_f$ formula [35]. The size of $\mathcal{A}_\varphi$ is therefore $O(2^{|\varphi|})$, where $|\varphi|$ is the size of the formula. From the NFA $\mathcal{A}_\varphi$, we construct a DFA $\mathcal{A}_{\mathsf{act}}$. The size of $\mathcal{A}_{\mathsf{act}}$ is the same of $\mathcal{A}_\varphi$, hence it remains $O(2^{|\varphi|})$. Then, we consider the size of the synchronous product $\mathcal{A}_{\varphi,\mathcal{C}}$, which combines $\mathcal{A}_{\mathsf{act}}$ with the state spaces of the services in the community $\mathcal{C}$, formed by n services $\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_n$. Each service has a state space of size $|\mathcal{S}_i|$. Therefore, the total size of the state space is $|Q| \cdot |\mathcal{S}_1| \cdot |\mathcal{S}_2| \cdots |\mathcal{S}_n|$, where $Q = O(2^{|\varphi|})$. Checking for non-emptiness of the DFA $\mathcal{A}_{\varphi,\mathcal{C}}$ is linear in the size of the automaton, which does not worsen the worst-case computational complexity cost. Finally, the overall size of $\mathcal{A}_{\varphi,\mathcal{C}}$ is exponential in both the size of the formula φ and the number of services n. However, the contribution of each service to the state space remains polynomial, as $|\mathcal{S}_i|$ is polynomial in the size of the individual services. □

Example 4. Continuing from Example 3, we have that $\mathcal{L}(\mathcal{A}_{\varphi,\mathcal{C}}) \neq \emptyset$. For example, consider the word $w = (clean, g_1, 1), (water, g_2, 2),$

$(pluck, g_4, 3), (empty, g_4, 3)$. It is easy to see that w is accepted by $\mathcal{A}_{\varphi,\mathcal{C}}$. Hence, by Theorem 1, we know the composition problem is realizable. In particular, note that, by Proposition 2, the corresponding history $h = (a_0, b_0, c_0) \xrightarrow{(clean,1)} (a_0, b_0, c_0) \xrightarrow{(water,2)} (a_0, b_0, c_0) \xrightarrow{(pluck,3)} (a_0, b_0, c_1) \xrightarrow{(empty,3)} (a_0, b_0, c_0)$ is successful, and the orchestrator that generates this execution is a solution to the composition problem.

6. Solution for goal-based LTL_f composition for nondeterministic services

Differently from Section 5, in this section we study the nondeterministic variant of our composition problem, i.e., when services can behave nondeterministically. A concrete instance of a nondeterministic service composition problem has been given in Example 1.

To synthesize the orchestrator, we rely on a different 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 DFA $\mathcal{A}_\varphi$; (2) compute a *product of the* DFA $\mathcal{A}_\varphi$ *with the services, obtaining a new* DFA $\mathcal{A}_{\varphi,\mathcal{C}}$; (3) the latter DFA can be seen as an arena over which we play the so-called DFA *game* [40]; (4) 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 DFA $\mathcal{A}_\varphi = (A, Q, q_0, F, \delta)$ of an LTL_f formula, which can be double-exponentially larger than the size of the formula, can be computed by exploiting a well-known correspondence between LTL_f formulas and automata on finite words [16].

Step 2. Then, given $\mathcal{A}_\varphi$ and $\mathcal{C}$, we build the *composition* DFA $\mathcal{A}_{\varphi,\mathcal{C}} = (A', Q', q_0', F', \delta')$ as follows:

- $A' = \{(a, i, \sigma_j) \mid (a, i, \sigma_j) \in A \times \{1, \dots, n\} \times \left(\bigcup_i \Sigma_i\right)$ and $\sigma_j \in \Sigma_i\}$;
- $Q' = Q \times \Sigma_1 \times \cdots \Sigma_n$;
- $q_0' = (q_0, \sigma_{10} \dots \sigma_{n0})$;
- $F' = F \times F_1 \times \cdots \times F_n$;
- $\delta'((q, \sigma_1 \dots \sigma_i \dots \sigma_n), (a, i, \sigma_i')) = (q', \sigma_1 \dots \sigma_i' \dots \sigma_n)$ iff $\sigma_i' \in \delta_i(\sigma_i, a)$, and $\delta(q, a) = q'$.

Intuitively, the DFA $\mathcal{A}_{\varphi,\mathcal{C}}$ is a synchronous cartesian product between the DFA $\mathcal{A}_\varphi$ and the service S_i chosen by the current symbol $(a, i, \sigma) \in A'$. 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 $(a_0, o_0, \sigma_{o_0}) \dots (a_m o_m, \sigma_{o_m}) \in A'^*$, which induces the run $r = (q_0, \sigma_{10} \dots \sigma_{n0}), \dots, (q_{m+1}, \sigma_{1,m+1} \dots \sigma_{o_j} \dots \sigma_{n,m+1})$ over $\mathcal{A}_{\varphi,\mathcal{C}}$, we define the history $h = \tau_{\varphi,\mathcal{C}}(w) = (\sigma_{10} \dots \sigma_{n0}), (a_0, o_0), \dots, (\sigma_{1,m+1} \dots \sigma_{n,m+1})$.

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 $w \in \mathcal{L}(\mathcal{A}_{\varphi,\mathcal{C}})$.*

Proof. We prove both directions of the equivalence separately.

($\Leftarrow$) Let w be a word $w = (a_0, o_0, \sigma_{o_0}) \dots (a_m, o_m, \sigma_{o_m}) \in A'^*$ Assume $w \in \mathcal{L}(\mathcal{A}_{\varphi,\mathcal{C}})$. We have that the induced run $r = (q_0, \sigma_{10} \dots \sigma_{n0}), \dots, (q_{m+1}, \sigma_{1,m+1} \dots \sigma_{n,m+1})$ over $\mathcal{A}_{\varphi,\mathcal{C}}$ is accepting. This means that $q_{m+1} \in F$ and $\sigma_{i,m+1} \in F_i$ for all i. Now consider the history $h = \tau_{\varphi,\mathcal{C}}(w) = (\sigma_{10} \dots \sigma_{n0}), (a_0, o_0), \dots, (a_m, o_m), (\sigma_{1,m+1} \dots \sigma_{n,m+1})$. For every $0 \le k \le m$, we have by definition of δ' that $\delta_i(\sigma_{i,k}, a_k) = \sigma_{i,k+1}$ if $i = o_k$, and $\sigma_{i,k} = \sigma_{i,k+1}$ otherwise. Finally, $o_k \in \{1 \dots n\}$, $a_k \in A$ and $\sigma_{i,k+1} \in \Sigma_i$, for all $i = 1, \dots, n$ and $0 \le k \le m$. Therefore, h is a valid history. Moreover, h is a successful history because $q_{m+1} \in F$ iff $a_0, \dots, a_m \vDash \varphi$, and $\sigma_{i,m+1} \in F_i$ for all i.

($\Rightarrow$) Assume that $h = (\sigma_{10} \dots \sigma_{n0}), (a_0, o_0), \dots, (\sigma_{1,m+1} \dots \sigma_{n,m+1})$ is a successful history with $\mathcal{C}$ Then we both have that (*i*) $\text{actions}(h) = a_0, \dots, a_m \vDash \varphi$ and (*ii*) $\sigma_{i,m+1} \in F_i$ for all i. By construction of the DFA $\mathcal{A}_\varphi$, proposition (*i*) implies that there exists a run $q_0, \dots, q_{m+1}$ where $q_{m+1} \in F$. Moreover, let $\sigma_1, \dots, \sigma_{m+1}$ be the sequence of service states s.t. $\sigma_k = \sigma_{i,k}$ (where $i = o_k$), for $1 \le k \le m+1$. Now consider the sequence $w = (a_0, o_0, \sigma_1), \dots, (a_m, o_m, \sigma_{m+1})$. By construction, we have $\tau_{\varphi,\mathcal{C}}(w) = h$. Finally, from proposition (*i*) and proposition (*ii*), it follows that $w \in \mathcal{L}(\mathcal{A}_{\varphi,\mathcal{C}})$. This concludes the proof. □

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. Now, 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 of $\mathcal{X}$ 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. The *specification* of the game is given by a DFA $\mathcal{G}$ whose alphabet is $\mathcal{X} \times \mathcal{Y}$. 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 kth 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 $Y_i = f(w_{\mathcal{X}}|_{i-1})$, 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 $s_0 \in Win(\mathcal{G})$ [40]. 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 $\delta_T(q, X) = \delta'(q, (X, \theta(q)))$, and $\theta_T : Q \to \mathcal{Y}$ is the output function defined as $\theta_T(q) = Y$ such that if $q \in Win_{i+1}(\mathcal{G}) \backslash Win_i(\mathcal{G})$ 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 $m \ge 0$ $w = (X_0, Y_0) \dots (X_m, Y_m)$, where for each index k, Y_k and X_k are of the form (a_k, 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 $(a_m, o_m) = \theta_T(\delta_T^*(q_0, w))$, and whenever the trace satisfies the successful condition, we have that the next choice is $\perp$.

In the following, we show the correctness of our approach, i.e. that 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 $\mathcal{Y} = A \times \{1, \dots, n\}$.

Theorem 3. *Let $\langle \varphi, \mathcal{C} \rangle$ be the nondeterministic composition problem for LTL_f goals. Realizability can be solved by checking whether $q_0' \in Win(\mathcal{A}_{\varphi,\mathcal{C}})$.*

Proof. The proof is rather technical, therefore we first give an intuitive explanation. 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 $q'_0 \notin Win(\mathcal{A}_{\varphi,\mathcal{C}})$; the latter implies that we can build an arbitrarily long history that is not successful, by definition of winning region, contradicting that γ realizes φ.

We now provide the full proof of the claim by separately proving the soundness and completeness of our approach.

Soundness. Assume $q'_0 = (q_0, \sigma_{10} \dots \sigma_{n0}) \in Win_m(\mathcal{A}_{\varphi,\mathcal{C}})$, i.e. the controller can win in at most m steps; we aim to show that there exists an orchestrator γ that realizes φ, i.e. all executions t are finite and successful.

We prove it 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.

Base case ($k = 0$): assume $q'_0 \in F'$, i.e. q'_0 is already a goal state. Then, the problem is trivially realizable since the goal specification is already satisfied (the execution $t = (\sigma_{10} \dots \sigma_{n0})$ is a successful execution for any γ).

Inductive case: assume the claim holds for every state $q'_k \in Q'$ that can reach an accepting state in at most k steps, i.e. $q'_k = (q_k, \sigma_{1k} \dots \sigma_{nk}) \in Win_k(\mathcal{A}_{\varphi,\mathcal{C}})$, and let γ_k be the orchestrator that realizes the goal specification starting from such states. Let $\Delta Win_{k+1}(\mathcal{A}_{\varphi,\mathcal{C}}) = Win_{k+1}(\mathcal{A}_{\varphi,\mathcal{C}}) \setminus Win_k(\mathcal{A}_{\varphi,\mathcal{C}})$. Consider a state $q'_{k+1} = (q_{k+1}, \sigma_{1,k+1} \dots \sigma_{n,k+1}) \in \Delta Win_{k+1}(\mathcal{A}_{\varphi,\mathcal{C}})$. By construction, $q'_{k+1} \in PreC(Win_k(\mathcal{A}_{\varphi,\mathcal{C}}))$. This means that there exist a controllable symbol $Y \in \mathcal{Y}$ such that for all uncontrollable symbols $X \in \mathcal{X}$ we can reach a state in $Win_k(\mathcal{A}_{\varphi,\mathcal{C}})$, i.e. $\delta(q'_{k+1}, (X, Y)) = q'_k \in Win_k(\mathcal{A}_{\varphi,\mathcal{C}})$. Let γ_{k+1} be the orchestrator that behaves as γ_k in states in $Win_k(\mathcal{A}_{\varphi,\mathcal{C}})$, and $\gamma_{k+1}((\sigma'_{1,k+1} \dots \sigma'_{n,k+1})) = (a_{k+1}, o_{k+1})$, where $Y = (a_{k+1}, q'_{k+2}, o_{k+1})$, for every $\sigma'_{1,k+1} \dots \sigma'_{n,k+1}$ such that $(q'_{k+1}, \sigma'_{1,k+1} \dots \sigma'_{n,k+1}) \in \Delta Win_{k+1}(\mathcal{A}_{\varphi,\mathcal{C}})$. By the inductive hypothesis, we have that all finite executions t_k of γ_k, starting from some $(\sigma_{1k} \dots \sigma_{nk}) \in \Delta Win_k(\mathcal{A}_{\varphi,\mathcal{C}})$, are successful finite executions. To prove our claim, we only need to show that all t_{k+1}, starting from some state in $\Delta Win_{k+1}(\mathcal{A}_{\varphi,\mathcal{C}})$, are also successful executions. If $|t_{k+1}| \leq k$, e.g. when the adversary behaves cooperatively, then it holds by inductive hypothesis. For executions of length $k+1$, this is the case because, by construction, we have $t_{k+1} = \sigma'_{z,k+1}, (a_{k+1}, o_{k+1}), t'$ for some execution t' of γ_k, some $(q'_{k+1}, \sigma'_{z,k+1}) \in \Delta Win_{k+1}(\mathcal{A}_{\varphi,\mathcal{C}})$, and $(a_{k+1}, o_{k+1}) = \gamma_{k+1}(\sigma'_{z,k+1})$. In other words, t_{k+1} is a valid finite execution of γ_{k+1}, and moreover, it is successful since t' is successful. Finally, we have that γ_m, by induction, is an orchestrator that can force the win of the game from q'_0.**Completeness.** By contradiction, assume there exists an orchestrator γ that realizes φ with community $\mathcal{C}$, but that $q'_0 \notin Win(\mathcal{A}_{\varphi,\mathcal{C}})$; we will exhibit a trace t that is at the same time an execution of γ and not successful, hence contradicting the fact that γ realizes φ. If $q'_0 \notin Win(\mathcal{A}_{\varphi,\mathcal{C}})$, then it means, by definition of $Win(\mathcal{A}_{\varphi,\mathcal{C}})$ and $PreC$, that for all $Y \in \mathcal{Y}$, there exist $X \in \mathcal{X}$ such that the successor state q'_1 is not in $Win(\mathcal{A}_{\varphi,\mathcal{C}})$. Since the same property holds for q'_1, we can recursively generate an arbitrarily long word $w = (X_0, Y_0) \dots (X_m, Y_m)$, for any choice of $Y_0 \dots Y_m$, such that $w \notin \mathcal{L}(\mathcal{A}_{\varphi,\mathcal{C}})$. Now consider the infinite execution $t = (\sigma_{10} \dots \sigma_{n0}), (a_0, o_0), (\sigma_{11} \dots \sigma_{n1}), \dots$, built as follows: for all $k \geq 0$, $(a_k, o_k) = \gamma((\sigma_{10} \dots \sigma_{n0}), \dots, (\sigma_{1k} \dots \sigma_{nk}))$, and for any $Y_k = (a_k, o_k)$, take $X_k = \sigma$ such that $w_k = (Y_0, X_0) \dots (Y_k, X_k) \notin \mathcal{L}(\mathcal{A}_{\varphi,\mathcal{C}})$ and $\delta_{o_k}(\sigma_{o_k,k}, a_k) = \sigma$, and set $\sigma_{o_k,k} = \sigma$. In other words, we consider any valid execution t of γ where the successor state of the chosen service is taken according to the winning environment strategy (which is losing for the agent). By construction, t is an infinite execution of γ. Moreover, by Proposition 3, no finite prefix of t is successful, because no finite prefix w_k is accepted by $\mathcal{A}_{\varphi,\mathcal{C}}$ and, by construction of $\mathcal{A}_{\varphi,\mathcal{C}}$, this means that it is always the case that either $\mathsf{actions}(h) \not\models \varphi$, or for some service S_i, $\sigma_{ik} \notin F_i$. Since we assumed that γ realizes φ with community $\mathcal{C}$, but we can construct an execution t of γ that is not successful, we reached a contradiction. □

Theorem 4. *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 $\mathcal{C}$.*

Proof.

By definition, γ_T realizes φ with $\mathcal{C}$ if all its executions are finite successful traces. By contradiction, assume this is not the case, and that there exists an infinite execution $t = (\sigma_{10} \dots \sigma_{n0}), (a_0, o_0), \dots$, for which all finite prefixes t' of t are such that $\mathsf{actions}(t') \not\models \varphi$ and $last(\mathsf{states}(t')) \notin F_1 \times \dots \times F_n$. Since the set of states $\Sigma_1 \times \dots \times \Sigma_n$ is finite, it must be the case that a service configuration $\sigma_1, \dots \sigma_n$ is visited more than once in t. Consider the corresponding infinite trace produced by the strategy implemented by T while playing the game $\mathcal{A}_{\varphi,\mathcal{C}}$: $\tau = (q_0, \sigma_{10} \dots \sigma_{n0}), (a_0, q_1, o_0, \sigma_{o_0}), \dots$. By construction of $\mathcal{A}_{\varphi,\mathcal{C}}$, it follows that there exists an environment move X that forces the agent to loop infinitely and never reach the goal states in $\mathcal{A}_{\varphi,\mathcal{C}}$. But this contradicts the fact that T is a winning strategy that forces the game to reach an accepting state. □

Considering the cost of each step above, we get the following upper bound for the worst-case computational cost.

Theorem 5. *Service composition for LTL$_f$ goals can be solved in at most double-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.*

Proof. The analysis is the same as the proof for Theorem 2, except that now the technique performs full determinization of the automaton, which in the worst-case has size $O(2^{2^{|\varphi|}})$. □

Example 5. Continuing from Example 1, we now apply the technique presented above, starting from the nondeterministic service community and the DFA of the goal formula, shown in Fig. 2. After constructing the composition DFA (we omit the representation of the DFA $\mathcal{A}_{\varphi,\mathcal{C}}$ due to its complexity), we have to solve the DFA game over it. The controllable variables of the game are $\mathcal{Y} = \{clean, pluck, water, empty\} \times \{1, 2, 3\}$, while the uncontrollable variables are $\mathcal{X} = \{a_0, a_1, b_0, b_1, b_2, c_0, c_1\}$. Fig. 5 shows a winning strategy for the DFA game over $\mathcal{A}_{\varphi,\mathcal{C}}$. The labels next to the nodes are the evaluation of the output function θ over that state, while the edge labels represent the environment's response. Intuitively, the strategy selects the sequence of actions *clean*, *water*, *pluck* in order to satisfy the LTL$_f$ task. However, executing the action *clean* from state a_0 of $\mathcal{B}_1$ might cause the service to end up in state a_1, which is not accepting. Therefore, the strategy must take into account this possibility by reading this information after it took the process action *clean*; then, after completing the LTL$_f$ task, the state of the service $\mathcal{B}_1$ can be restored to a_0 via action *empty*.

Discussion. An interesting feature of the technique described in Section 5 is that it avoids the full determinization of the NFA $\mathcal{A}_\varphi$, saving an exponential blow-up in the overall computational complexity cost. Yet, precisely because of this optimization, the technique cannot adapted to be used as-is for the nondeterministic case. Intuitively, this happens because a construction that is analogous to $\mathcal{A}_{\varphi,\mathcal{C}}$, with the next service state in the alphabet, would mix the "existential" nondeterminism of the NFA of the goal formula (at least one run must be accepting) and the "universal" nondeterminism of the services' behavior (all runs implied by the services' nondeterminism must be handled successfully). To illustrate this, we exhibit a counterexample that shows how the technique can fail in the case of nondeterministic services:

Example 6. Let $\varphi = \bigcirc a \vee \bigcirc b$, and let $\mathcal{C} = \{S_1\}$, with S_1 being the unique nondeterministic service in the community, depicted in the right side of Fig. 6. Let $\mathcal{A}_\varphi$, shown on the left side of Fig. 6, be an NFA corresponding to φ. The goal specification φ is realizable with community $\mathcal{C}$. First, the orchestrator chooses action $(a, 1)$. The next

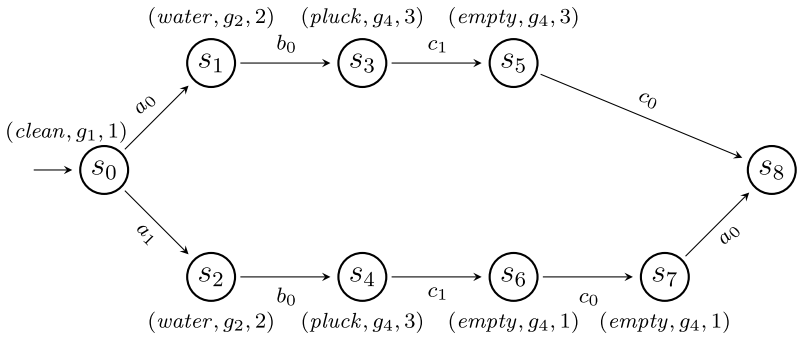


Fig. 5. A possible winning strategy for the DFA game over the composition DFA (Example 5). The label on the transducer's states is the chosen move for that state, while the label on the edges is the environment's response. The fork in s_0 is due to the nondeterminism of executing *clean* from state a_0 of $\mathcal{B}_1$. For this reason, the winning strategy must remember that service $\mathcal{B}_1$ has to be restored to the accepting state a_0 after completing the LTL$_f$ task.

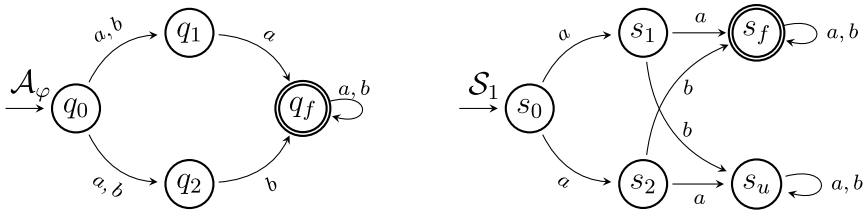


Fig. 6. Counterexample that shows that, in the nondeterministic case, using the NFA $\mathcal{A}_\varphi$ in the automata-theoretic construction does not work. On the left, the NFA $\mathcal{A}_\varphi$ corresponding to the goal formula $\varphi = \bigcirc a \vee \bigcirc b$; on the right, the service $\mathcal{S}_1$.

service state can nondeterministically be either s_1 or s_2. In case it is s_1, the orchestrator chooses action $(a, 1)$, which deterministically makes $\mathcal{S}_1$ to transition into s_f; otherwise, if the execution ends up in s_2, then the orchestrator chooses action $(b, 1)$, which makes $\mathcal{S}_1$ to transition into s_f. In both cases, the execution ends up in s_f, which is accepting. Since the goal specification required either a or b to be executed in the second step, the specification is satisfied. Moreover, since the final state of the execution is s_f, which is accepting, for both the possible finite executions $h_1 = s_0, (a, 1), s_1, (a, 1), s_f$ and $h_2 = s_0, (a, 1), s_2, (b, 1), s_f$, we conclude that γ realizes the specification φ with C.

Now, let us try to apply the solution technique presented in Section 5, designed for the deterministic case. Consider a possible strategy to win the DFA game $\mathcal{A}_{\varphi,C}$. If the first move of the protagonist strategy is $(aq_1, 1)$, then the adversary can transition the service into s_2. From there, we have that the strategy cannot take the service action b, whose effect would be to move to the service's accepting state s_f, since this is not allowed in state q_1 of $\mathcal{A}_\varphi$; instead, the protagonist is forced to take the only allowed action $(a, q_f, 1)$, which leads to the failure state s_u of $\mathcal{S}_1$. Therefore, this strategy is not winning. On the other hand, let the protagonist agent's first move be $(a, q_2, 1)$. Then, the adversary can transition the service into s_1. From there, the protagonist agent cannot take the service action a, whose effect would be to move to the service's accepting state s_f; instead, he is forced to take action $(b, q_f, 1)$, whose effect on the service component of the state is to move into s_u. Therefore, since we exhaustively enumerated all possible strategies of the protagonist agent and found out that all of them are not winning, we conclude that the dfa game cannot be won. Therefore, we erroneously conclude that the specification is not realizable.

7. Reduction to planning

The theoretical analysis of Section 5 and Section 6 does not automatically translate into a practical and efficient technique to solve the composition problem. To do so, in this section, we present a reduction from our composition problem $\langle \varphi, C \rangle$ into a corresponding planning problem, which allows us to leverage decades of efforts in research and engineering of efficient and performant automated planners. We handle the deterministic and the nondeterministic cases in two separate subsections.

7.1. Torres and baier's construction for the deterministic case

In this section, we show how we can take advantage of the NFA-based construction to solve the deterministic case of the goal-based LTL$_f$ service composition problem. Specifically, we show how we can compute a PDDL specification that is *linear* in the size of the formula φ, by exploiting Torres and Baier's technique for planning for LTL$_f$ goals [24]. 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. 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. Moreover, we are still in the context of classical planning since, in this section, we are considering deterministic services.

The automata-theoretic solution technique presented in Section 5 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 planning domain can be exponential in the size of the formula and exponential in the number of services, meaning that computing the entire domain beforehand may be infeasible. However, this is not required since we can build the domain on-the-fly while searching for a solution. We can do this by leveraging on planning where the agent controls the actions and the domain simulates the execution of the action in the service community [6]. In particular, we focus on planning for temporally extended goals [42,43], works on declarative and procedural constraints [44–46], and temporal logics with finite-trace semantics, such as LTL$_f$ [16,24,45,46].

In our setting, the orchestrator controls the actions to satisfy the LTL$_f$ goal. A forward search-based approach is particularly interesting for our case since it could possibly avoid the exponential construction of the entire domain as the procedure stops as soon as a successful plan 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 [24]; more details later in this section. Another crucial feature of the automated planning framework 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 domain (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. $Q_N \subseteq 2^{Q_A}$ [16,24]. As for the services 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.

Most of the techniques in 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) 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 [24] 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: [46]'s translation is worst-case exponential but builds the entire NFA explicitly and only works for deterministic domains; and the encoding proposed in [18] requires the construction of the entire DFA, which is not optimal since its size is doubly exponential in the size of the formula.

First, we define the *(deterministic) service community domain* $\mathcal{D}^d_{\mathcal{C}} = \langle \mathcal{F}', A', pre', eff' \rangle$ and *problem* $\Gamma^d_{\mathcal{C}} = \langle \mathcal{D}^d_{\mathcal{C}}, s'_0, G' \rangle$, where:

- $\mathcal{F}' = \{\textbf{start}\} \cup A \cup \Sigma_1 \cup \cdots \Sigma_n$;
- $A' = A \times \{1 \dots n\}$;
- $pre'(\langle a, i\rangle) = true$ (since δ_i are total functions);
- $eff'(\langle a, i\rangle) = eff'_{a,i}(\sigma_{i1}, \sigma_{i2})$, where $\sigma_{i2} = \delta_i(\sigma_{i1}, a)$ and $eff'_{a,i}(\sigma_{i1}, \sigma_{i2}) = \{\{\sigma_{i1}\} \rhd \{\neg\sigma_{i1}, \sigma_{i2}, a\} \cup \bar{A}(a)\}$, with $\bar{A}(a) = \{\neg a' \mid a' \in A, a' \neq a\}$;
- $s'_0 = \{\textbf{start}\} \cup \{\sigma_{i0} \mid \sigma_{i0}$ is the initial state of $S_i\}$;
- $G' = \{(\sigma_1, \dots, \sigma_n) \mid \sigma_i \in F_i$ for all $i = 1, \dots n\}$.

Intuitively, $\mathcal{D}^d_{\mathcal{C}}$ 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 A-component in the state is needed because the formulas can only be evaluated over state fluents and not over planning actions. 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 unnecessary (though conditional effects are) since we assumed the services' transition functions are complete. The effect of action (a, i) is defined as follows: (*i*) removes the current service state σ_{i1}, (*ii*) adds the successor state $\sigma_{i2} = \delta_i(\sigma_{i1}, a)$, (*iii*) adds the new action a, and (*iv*) removes all other actions. 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 $\bar{A}(a)$, hence forcing the simple trace semantics. 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 [24]. Given a problem instance $\langle \varphi, \mathcal{C}\rangle$, and given the domain $\mathcal{D}^d_{\mathcal{C}}$ and problem $\Gamma_{\mathcal{C}}$ as defined above, we get a new domain $\mathcal{D}^d_{\varphi,\mathcal{C}}$ and $\Gamma^d_{\varphi,\mathcal{C}}$ by applying the translation rules with the formula $\varphi' = \textbf{start} \wedge \bigcirc\varphi$. Intuitively, the evaluation of φ' will read the initial state and, from there on, will evaluate the chosen actions, which, by constructions, are added to the subsequent states. The new goal states, which encode the acceptance of the LTL$_f$ goal, must also include the acceptance condition on the service's states as in G'.

Theorem 6. *Let $\langle \varphi, \mathcal{C}\rangle$ be an instance of LTL$_f$-goal oriented service composition with deterministic services. φ is realizable with $\mathcal{C}$ iff there exists a strong solution for the deterministic planning problem $\Gamma^d_{\varphi,\mathcal{C}}$.*

Proof (*Proof sketch*)**.** By construction, and by correctness of Torres and Baier's construction, there is a one-to-one correspondence between traces of $\Gamma^d_{\varphi,\mathcal{C}}$ and the composition DFA $\mathcal{A}_{\varphi,\mathcal{C}}$, modulo deletion of the first evaluation of **start** and the auxiliary synchronization actions of the Torres and Baier's construction. In other words, the search graph induced by $\mathcal{D}^d_{\varphi,\mathcal{C}}$ and $\Gamma^d_{\varphi,\mathcal{C}}$ is essentially the same of the one induced by $\mathcal{A}_{\varphi,\mathcal{C}}$. Therefore, there exists a plan for $\Gamma^d_{\varphi,\mathcal{C}}$ iff there exists an accepting trace for $\mathcal{A}_{\varphi,\mathcal{C}}$. Then, the claim follows by Theorem 1. □

By inspection, we can see that we do maintain the computational cost stated in the previous section in the worst case.

Theorem 7. *LTL$_f$ goal-oriented service composition in the deterministic variant 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 compact, i.e. logarithmic, form).*

We observe that in the construction of $\mathcal{D}^d_{\varphi,\mathcal{C}}$ we introduce action fluents, increasing the state space's size. This could be avoided by modifying Torres and Baier's encoding with PDDL3 action-trajectory constraints [47], and hence by evaluating the temporal goal over the action-trajectory only. A detailed analysis of this option is left as future work.

Example 7. Starting from the deterministic Garden Bots scenario described in Example 2, we show

7.2. Encoding in adversarial FOND planning for the nondeterministic case

To solve the nondeterministic variant of the goal-based LTL$_f$ service composition problem, we cannot rely on a classical planning technique (i.e., for deterministic domain only) since, from the point of view of the agent/orchestrator, there might be multiple successor states after an action execution. For this purpose, we can leverage FOND adversarial planning (aka strong planning in FOND), where the agent controls the actions and the environment controls the fluents [6].

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. This is not required since we can build the arena on the fly while searching for a solution. We can do this by reducing the problem to a FOND adversarial planning (aka strong planning in FOND) problem, where the agent controls the actions and the environment controls the fluents [6]. In particular, we focus on FOND planning for temporally extended goals [17,18,40]. More recently, [48,49] proposed, for classical and FOND planning, respectively, a polynomial-time compilation in PDDL for goals in Pure-Past Linear Temporal Logic (PPLTL) [50]. In our implementation, we will use the encoding designed for strong LTL$_f$ FOND planning from Camacho and McIlraith [25], but different techniques can be used.

In this 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. Therefore, a forward search-based approach able to handle environment nondeterminism is particularly interesting for our case because it could possibly avoid the exponential construction of the entire arena since the procedure stops as soon as a winning strategy is found. Alas, for LTL$_f$, it is not possible to use a linear representation since a full determinization requires a double-exponential blow-up in the state space of the DFA. The factorized PDDL representation can only "save" an exponential in the optimal case. In fact, many techniques for FOND planning for LTL$_f$ goal first compute the DFA and then encode it in the PDDL domain, using either a linear or symbolic representation for its state space, and its execution is simulated alongside the original planning domain, according to the actions taken by the agent.

The definitions of $\mathcal{D}_C^{nd}$ and Γ_C^{nd} are the same as the deterministic case, except that the function eff of $\mathcal{D}_C^{nd}$ now must take into account nondeterministic effects due to the services' nondeterministic behavior. In particular: $eff'(\langle a, i\rangle) = \{eff'_{a,i}(\sigma_{i1}, \sigma_{i2}) \mid \sigma_{i2} \in \delta_i(\sigma_{i1}, a)\}$ where $eff'_{a,i}(\sigma_{i1}, \sigma_{i2}) = \{\{\sigma_{i1}\} \rhd \{\neg\sigma_{i1}, \sigma_{i2}, a\} \cup \bar{A}(a)\}$. Intuitively, now the effects 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 $eff'_{a,i}(\sigma_{i1}, \sigma_{i2})$). Finally, the planning problem combined with the goal formula φ considers the cross-product between the DFA of φ, $\mathcal{A}_\varphi$, and the nondeterministic service community domain $\mathcal{D}_C^{nd}$, built using Camacho and McIlraith's technique [25].

Theorem 8. *Let $\langle \varphi, \mathcal{C}\rangle$ be an instance of LTL$_f$-goal oriented service composition in the nondeterministic case. φ is realizable with $\mathcal{C}$ iff there exists a strong solution for the FOND adversarial problem $\Gamma_{\varphi,\mathcal{C}}$.*

Proof (*Proof sketch*)**.** By the correctness of the construction and the correctness of Camacho and McIlraith'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** and auxiliary actions for synchronization between the domain and the DFA states). In other words, the game arena induced by $D_{\varphi,\mathcal{C}}$ and $\Gamma_{\varphi,\mathcal{C}}$ is essentially the same of the one induced by $\mathcal{A}_{\varphi,\mathcal{C}}$. Therefore, there exists a strong solution for $\Gamma_{\varphi,\mathcal{C}}$ iff there exists a winning strategy for the DFA game over $\mathcal{A}_{\varphi,\mathcal{C}}$. The claim follows by Theorem 4. □

By inspection, we can see that we do maintain the computational cost stated in the previous section in the worst case.

Theorem 9. *LTL$_f$ goal-oriented service composition in the nondeterministic can be solved in at most double-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 compact, i.e. logarithmic, form).*

8. Implementation and applications

This section describes our software prototype to solve the composition problem and the tests we executed on industrial case studies taken from the literature. The code can be found on GitHub.[5]

8.1. 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 planning task, using the technique formalized in the previous sections. First, we construct $\mathcal{D}_C$ and Γ_C in PDDL form. The PDDL domain file models the (possibly) nondeterministic behavior 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 potential successors. To workaround this issue, given an action $\langle a, i\rangle$ of $\mathcal{D}_C$, 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.

There are different options depending on whether the service community is deterministic or nondeterministic. In the deterministic case, the tool can use either Torres and Baier's translator [24] (in the following, denoted with TB), or Camacho and McIlraith's translator ltlfond2fond [25]. Observe that the former option is optimal with respect to the problem's computational complexity, while the latter requires the full determinization of the goal formula's DFA, hence with doubly exponential cost. In the nondeterministic case, we cannot rely on the TB translator since the nondeterminism of the services' behavior requires the full determinization of the goal automaton (see discussion of Example 6).

The TB translator, implemented in SWI-Prolog, works by encoding the on-the-fly evaluation of the LTL$_f$ goal specification φ in the domain behavior. The TB translator supports four modes: Simple, OSA, PG, and OSA+PG, where Simple is the "naive" translation (cfr. [24], Section 4) and OSA, OSA+PG are two optimizations called "Order for Synchronization Action" and "Positive Goals" (cfr. *ibid.*, Section 4). OSA+PG is the combination of OSA and PG. The encoding of the goal formula in PDDL follows the syntax supported by the TB translator, which we recall now. The goal LTL$_f$ formula φ is encoded in PDDL using the following translation function $\mathcal{T}$:

- $\mathcal{T}(a) =$ (a): the atomic proposition is encoded as an atomic PDDL predicate. E.g. the action *move_up* is translated into the atomic ground predicate (move_up);
- $\mathcal{T}(\neg\varphi) =$ (not $\mathcal{T}(\varphi)$);
- $\mathcal{T}(\varphi_1 \wedge \varphi_2) =$ (and $\mathcal{T}(\varphi_1)$ $\mathcal{T}(\varphi_2)$);
- $\mathcal{T}(\bigcirc\varphi) =$ (next $\mathcal{T}(\varphi)$);
- $\mathcal{T}(\bullet\varphi) =$ (weaknext $\mathcal{T}(\varphi)$);
- $\mathcal{T}(\varphi_1 \,\mathcal{U}\, \varphi_2) =$ (until $\mathcal{T}(\varphi_1)$ $\mathcal{T}(\varphi_2)$);
- $\mathcal{T}(\varphi_1 \,\mathcal{R}\, \varphi_2) =$ (release $\mathcal{T}(\varphi_1)$ $\mathcal{T}(\varphi_2)$);
- $\mathcal{T}(\Diamond\varphi) =$ (eventually $\mathcal{T}(\varphi)$);
- $\mathcal{T}(\Box\varphi) =$ (always $\mathcal{T}(\varphi)$).

The final `(:goal)` section of the PDDL problem file includes, in conjunction: (*i*) the goal specified by the TB encoding, and (*ii*) the formula that specifies the accepting configuration for all services. Regarding (*ii*), the PDDL formula for the accepting service configuration has the form:

```
(and
  (or (curstate_s1 σ11)(curstate_s1 σ12)...)
  ...
  (or (curstate_sn σn1)(curstate_sn σn2)...)
)
```

For all services S_i with $i = 1 \ldots n$, 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.). The TB translator supports four modes: Simple, OSA, PG, and OSA+PG, where

[5] https://github.com/luusi/ltlf-goal-oriented-service-composition/

```
(define (domain composition)
  (:requirements :strips :typing
   :non-deterministic :conditional-effects)
  (:types state action)
  (:constants
    s0_a0 - state s1_b0 - state
    s2_c0 - state s2_c1 - state)
  (:predicates
    (curstate_0 ?s - state)
    (curstate_1 ?s - state)
    (curstate_2 ?s - state)
    (clean) (empty) (pluck) (water) (start_aux))
  (:action start_action_aux
    :precondition (start_aux)
    :effect (not (start_aux)))
  (:action clean_0_a0
    :precondition (and (not (start_aux)) (curstate_0 s0_a0))
    :effect (and (curstate_0 s0_a0)
      (clean) (not (empty)) (not (pluck)) (not (water))))
  (:action empty_2_c1
    :precondition (and (not (start_aux)) (curstate_2 s2_c1))
    :effect (and (not (curstate_2 s2_c1)) (curstate_2 s2_c0)
      (empty) (not (clean)) (not (pluck)) (not (water))))
  (:action pluck_2_c0
    :precondition (and (not (start_aux)) (curstate_2 s2_c0))
    :effect (and (not (curstate_2 s2_c0)) (curstate_2 s2_c1)
      (pluck) (not (clean)) (not (empty)) (not (water))))
  (:action water_1_b0
    :precondition (and (not (start_aux)) (curstate_1 s1_b0))
    :effect (and (curstate_1 s1_b0)
      (water) (not (clean)) (not (empty)) (not (pluck))))
)
```

```
(define (problem service-problem)
  (:domain composition)
  (:init
    (curstate_0 s0_a0)
    (curstate_1 s1_b0)
    (curstate_2 s2_c0)
    (start_aux))
  (:goal
    (and
      (start_aux)
      (next
        (and
          (clean)
          (next
            (until
              (clean)
              (or
                (and water (next pluck))
                (and pluck (next water))))))))))
)
```

Fig. 7. PDDL domain and problem files generated by executing our compilation tool on the deterministic Garden Bots scenario (Example 2).

Simple is the "naive" translation (cfr. [24], Section 4) and OSA, OSA+PG are two optimizations called "Order for Synchronization Action" and "Positive Goals" (cfr. *ibid.*, Section 4). OSA+PG is the combination of OSA and PG.

For explanatory purposes, in Fig. 7, we give an example of the PDDL domain and problem files generated from the deterministic Garden Bots scenario before applying the TB encoding.

8.2. 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 [51]. We consider the production process of an electric motor widely used in various applications such as industrial machinery, electric vehicles, household appliances, and many others [52]. To function properly, electric motors require certain materials that possess specific electrical and magnetic properties. Therefore, before the manufacturing processes start, the raw materials (i.e., copper, steel, aluminum, magnets, insulation materials, bearings) must be extracted and refined to obtain essential metals and polymers for electric motor parts manufacturing. When the materials are in the manufacturing facility, the effective manufacturing process can start. For the sake of brevity, in the following, we focus on the main aspects of the manufacturing process, skipping the provisioning, but the formalization can be easily extended to cover more details.

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:

$$\Diamond(assembleMotor) \tag{1}$$

$$\wedge\,(\neg assembleMotor\;\mathcal{U}\;buildStator) \tag{2}$$

$$\wedge\,(\neg assembleMotor\;\mathcal{U}\;buildRotor) \tag{3}$$

$$\wedge\,(\neg assembleMotor\;\mathcal{U}\;buildInverter) \tag{4}$$

$$\wedge\,\Diamond(staticTest \vee electricTest) \tag{5}$$

$$\wedge\,(\neg electricTest\;\mathcal{U}\;assembleMotor) \tag{6}$$

$$\wedge\,(\neg staticTest\;\mathcal{U}\;assembleMotor) \tag{7}$$

The $\mathcal{U}$-formulas (2), (3) and (4) prevent the assembly step before all the components are available, and the final goal is specified by $\Diamond(assembleMotor)$(1). Formula (5) specifies that either $staticTest$ or $electricTest$ must be executed. Finally, formulas (6) and (7) specify that the tests must be executed after the assembly step. Note that we could omit the DECLARE conditions in the formula since they are forced at the semantic level. In this scenario, the services can be considered machines that produce specific components or human operators that perform the tests manually. We consider two types of services: *infallible* and *breakable* (Fig. 8). 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:

$$\Diamond(cleaning \wedge \neg filmDeposition \wedge \neg \cdots$$
$$\Diamond(filmDeposition \wedge \neg \cdots$$
$$\Diamond(resistCoating \wedge \neg \cdots$$
$$\Diamond(exposure \wedge \neg \cdots$$
$$\Diamond(development \wedge \neg \cdots$$
$$\Diamond(etching \wedge \neg \cdots$$
$$\Diamond(impuritiesImplantation \wedge \neg \cdots$$

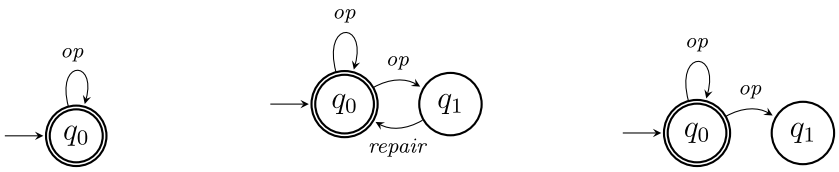


Fig. 8. The infallible, breakable and irreparable services templates, respectively.

$$\Diamond(activation \wedge \neg \cdots$$
$$\Diamond(resistStripping \wedge \neg \cdots$$
$$\Diamond(assembly \wedge \neg \cdots$$
$$\Diamond(testing \wedge \neg \cdots$$
$$\Diamond(packaging \wedge \neg \cdots)))))))))))$$

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.

8.3. Evaluation

We evaluated the Fast Downward planner [53] for evaluating the deterministic instances, and the MyND Planner [54] for the nondeterministic instances, combined with the h_{ff} and $h_{\max}$ heuristics, over the PDDL files produced by our tool. For the deterministic instances, we considered both the TB translator with the 4 available encoding types, and the ltlfond2fond translator, while for the nondeterministic instances, we only considered the ltlfond2fond 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: i_0 is simply the Garden Bots System Example 1; e_i, with $i = 0, \ldots, 6$, is the electric motor scenario with i builder services breakable and $6 - i$ infallible; c_i are the instances on the chip production scenario, with $i = 1, \ldots, 12$ 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 s ($\approx$ 15 min).

8.4. 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 64 GB of RAM. The JVM version is 14 for compatibility with MyND. The maximum RAM for the JVM was 16 GB.

8.5. Results

We report the results for the deterministic and nondeterministic cases separately.

Deterministic scenarios. The results of our evaluation on the deterministic variant of the Chip Production scenario are shown in Table 2, while the results for the deterministic Electric Motor scenario are shown in Table 1. Moreover, we also show the plots for the PT metric for the Chip Production scenario in Fig. 9. In both scenarios, at a high level, we observe that the PT metric when using the ltlfond2fond encoding is generally lower than when using the PT encoding, together with a lower EN and PS metric for the h_{ff}. This can be explained by the fact that, despite the TB encoding being worst-case optimal from a computational complexity point of view, very often the minimal DFA of the goal formula is of manageable size, hence allowing a simpler PDDL compilation and, in turn, a more compact search graph. On the other hand, the TB encoding adds much more overhead, with many of the visited AFA states being semantically redundant. Similar considerations can be made for the EN and PS metrics: the procedure based on ltlfond2fond encoding has a much lower number of expanded nodes and a much shorter plan length than the TB-based results, regardless of the action mode. Regarding the TT metric, we have that the results for the Simple encoding and the OSA encoding were the lowest for all instances compared to PG and OSA+PG. Notably, the PG and OSA configurations were the only configurations whose executions timed out.

By focusing on the comparison across different action modes of the TB encoding, 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. Regarding the EN metric, we observe mixed results: no approach dominates the other in all cases, although we observe that the EN in the OSA experiments for large instances was smaller than the other cases, while the PG encoding has a smaller number of EN for simpler instances. Finally, the measurements for the PS metric were the smallest with the PG encoding for both heuristics.

An interesting conclusion we draw is that, despite the TB encoding being worst-case optimal with respect to the size of the NFA, and so of the overall complexity, the configurations based on the TB encoding performed much worse than the configurations based on the ltlfond2fond encoding, especially on the EN and PT metrics for larger instances. We attribute this different behavior to a well-known empirical observation in the implementation of automata-theoretic techniques: that the determinization and minimization of automata on finite words is often more convenient than working directly with the nondeterministic automata [55]. Hence, being the DFA computed by ltlfond2fond more manageable and smaller in size so it is also the overhead in the compiled PDDL in terms of the number of auxiliary symbols and actions to evaluate the DFA transitions. We do not exclude, however, that in cases where the goal specification is particularly complex, it is more convenient to try to construct the NFA on-the-fly as needed instead of the full DFA construction, especially if the sequence of actions to reach a goal state is relatively short. On the other hand, for complex formulas, the number of auxiliary actions and symbols increases, as also witnessed by the PS metric (i.e., the length of the plan). Another result we can draw is that there are no noticeable differences in the performance between the usage of h_{ff} and h_{hmax}, especially for the TT metric. This could be due to either the scenarios being too easy to make a relevant difference emerge, or because both heuristics behave similarly due to the particular structure of our planning problem. Nevertheless, regarding the EN metric, we can observe that the h_{ff} is better almost by a factor of two. The design of domain-specific planning heuristics for our PDDL compilation would be an interesting future research.

Nondeterministic scenarios. In the nondeterministic Chip Production scenario (Fig. 10(b) and Table 4, left), the performance was quite worse than the deterministic case for all encodings and heuristics; the executions with heuristic $h_{\max}$ and h_{ff} timed out from cn7 and cn8, respectively. Generally, the running time were slightly lower with the heuristic h_{ff}. In the unsolvable Chip Production scenario (Fig. 10(c) and Table 4, right), the OSA action mode was considerably better than the other encodings, with comparable performance between $h_{\max}$ and h_{ff}. For all Chip production variants, the TT metric was always lower than 1 s for Simple and OSA, and for PG and OSA+PG it was increasing with the length of the formula by a factor of 2 at each increment. In the Electric Motor scenario (Fig. 10(a) and Table 3, left), similarly to the

Table 1
Evaluation metrics over the deterministic Electric Motor scenario, using the Fast Downward planner with $h_{\max}$ and h_{ff} heuristics, both for the TB and ltlfond2fond encodings.

Electric motor scenario (deterministic, TB)																
	Simple				OSA				PG				OSA+PG			
	TT	PT	EN	PS	TT	PT	EN	PS	TT	PT	EN	PS	TT	PT	EN	PS
$h_{\max}$																
e0	**0.2362**	12.041	30854	83	0.2437	**9.929**	**20962**	210	0.2482	12.481	30562	**80**	0.2574	11.599	320909	201
h_{ff}																
e0	0.252	12.204	30775	83	**0.2401**	**9.805**	**20962**	210	0.2509	12.514	30200	**80**	0.2584	12.233	321479	201

Electric motor scenario, (deterministic, ltlfond2fond)								
	$h_{\max}$				h_{ff}			
	TT	PT	EN	PS	TT	PT	EN	PS
e0	0.1162	**0.479**	1929	**31**	0.1319	0.488	**1775**	**31**

Table 2
Evaluation metrics over the Chip Production scenario, using MyND with $h_{\max}$ and h_{ff} heuristics.

Chip Production scenario (deterministic, TB)																
	Simple				OSA				PG				OSA+PG			
	TT	PT	EN	PS	TT	PT	EN	PS	TT	PT	EN	PS	TT	PT	EN	PS
$h_{\max}$																
c1	**0.2189**	**0.344**	21	17	0.2191	5.956	30	27	0.2314	0.363	**16**	**14**	0.2303	7.623	38	24
c2	0.2363	7.952	77	29	**0.2286**	**7.758**	127	64	0.24	7.782	**67**	**26**	0.2327	7.82	240	60
c3	**0.2351**	10.082	214	45	0.238	**9.177**	406	120	0.2585	10.285	**201**	**42**	0.2574	9.296	1111	115
c4	**0.2484**	14.788	539	65	0.2528	**10.841**	1026	204	0.2709	14.724	**523**	**62**	0.2808	10.883	4570	200
c5	**0.2535**	18.592	1288	89	0.2731	**13.684**	2335	322	0.3435	18.901	**1268**	**86**	0.3411	13.931	13103	318
c6	**0.2697**	25.387	2997	117	0.2805	**15.695**	4798	480	0.4702	25.509	**2973**	**114**	0.4916	16.053	30968	475
c7	**0.2939**	28.489	6866	149	0.319	**19.745**	9096	684	0.8094	27.616	**6856**	**146**	0.8087	20.057	64207	676
c8	**0.3372**	32.3	15551	185	0.3772	**24.667**	**15283**	940	1.4207	32.275	15540	**182**	1.442	26.187	123579	937
c9	**0.4341**	38.679	34876	225	0.4626	**31.52**	**25909**	1254	2.8202	39.754	34864	**222**	2.7203	36.435	216972	1243
c10	**0.5692**	49.64	77513	269	0.6107	**38.824**	**39583**	1632	5.0742	50.746	77500	**266**	5.0884	48.403	368869	1628
c11	**0.8339**	55.331	170854	317	0.8949	**48.374**	**61028**	2080	9.2818	60.018	170840	**314**	9.3006	66.468	586928	2066
c12	**1.0428**	68.391	373779	**369**	1.0516	**57.005**	**88570**	2604	–	–	–	–	–	–	–	–
h_{ff}																
c1	**0.2141**	**0.343**	20	17	0.2237	5.947	30	27	0.2239	0.373	**16**	**14**	0.2258	7.579	38	24
c2	**0.2313**	7.899	74	29	0.2361	7.818	127	64	0.2326	7.923	**56**	**26**	0.246	**7.791**	252	60
c3	**0.2469**	9.959	204	45	0.2518	**9.07**	406	120	0.2544	10.459	**179**	**42**	0.2615	9.228	1111	115
c4	**0.2373**	14.731	514	65	0.2447	**10.677**	1026	204	0.2617	14.703	**475**	**62**	0.2708	10.924	4570	200
c5	**0.2518**	18.616	1232	89	0.2616	**13.848**	2335	322	0.3339	19.157	**1171**	**86**	0.3328	14.019	13092	318
c6	**0.2756**	25.287	2878	117	0.2797	**15.808**	4798	480	0.4727	25.608	**2779**	**114**	0.514	16.174	30968	475
c7	**0.2946**	28.311	6620	149	0.317	**19.759**	9096	684	0.8121	27.698	**6451**	**146**	0.8018	20.711	64207	676
c8	0.3721	31.89	15050	185	**0.3707**	**24.793**	15283	940	1.4416	32.339	**14747**	**182**	1.441	27.59	123579	937
c9	**0.4282**	38.386	33864	225	0.4619	**31.888**	**25909**	1254	2.8172	40.089	33299	**222**	2.7066	38.835	217012	1243
c10	**0.5667**	49.522	75478	269	0.6119	**39.077**	**39583**	1632	5.087	51.346	74395	**266**	5.0775	54.834	368869	1628
c11	**0.8313**	55.921	166772	317	0.8861	**48.56**	**61028**	2080	9.3864	63.026	164659	**314**	9.3125	78.449	586901	2066
c12	**1.0449**	67.974	365602	**369**	1.0512	**58.27**	**88570**	2604	–	–	–	–	–	–	–	–

Chip Production scenario (deterministic, ltlfond2fond)								
	$h_{\max}$				h_{ff}			
	TT	PT	EN	PS	TT	PT	EN	PS
c1	0.107	0.147	**19**	**11**	0.115	**0.137**	**19**	**11**
c2	0.1128	**0.139**	**26**	**16**	0.1116	0.149	27	**16**
c3	0.1056	**0.157**	57	**21**	0.1124	0.167	**46**	**21**
c4	0.1173	**0.193**	101	**26**	0.1272	0.222	**82**	**26**
c5	0.1306	**0.272**	214	**31**	0.1222	0.291	**148**	**31**
c6	0.1277	**0.486**	379	**36**	0.1412	0.495	**266**	**36**
c7	0.1478	**0.974**	705	**41**	0.1437	0.985	**462**	**41**
c8	0.1709	2.238	1213	**46**	0.1617	**2.217**	**624**	**46**
c9	0.203	5.11	2017	**51**	0.1914	**5.106**	**1285**	**51**
c10	0.2428	12.236	3324	**56**	0.2486	**12.01**	**2007**	**56**
c11	0.3395	**14.275**	5256	**61**	0.3295	14.287	**2983**	**61**
c12	0.5075	**13.987**	8306	**66**	0.489	13.994	**4411**	**66**

Table 3
Evaluation metrics over the nondeterministic Electric Motor scenario, both solvable (left) and unsolvable (right) variants, using the ltlfond2fond encoding and MyND with $h_{\max}$ and h_{ff} heuristics.

Electric Motor scenario (nondeterministic)								
	$h_{\max}$				h_{ff}			
	TT	PT	EN	PS	TT	PT	EN	PS
e0	0.1093	0.406	1901	**31**	0.1344	**0.381**	**1654**	**31**
e1	0.1313	**0.954**	6148	**46**	0.1352	**0.954**	**4100**	**46**
e2	0.1362	3.81	27969	71	0.1347	**1.884**	**12899**	**51**
e3	0.1352	**6.55**	**39002**	96	0.1323	8.307	44226	**71**
e4	0.1168	94.901	**123724**	267	0.1217	**91.956**	136319	**217**
e5	0.117	203.227	170045	227	0.1172	**125.217**	**145677**	**182**
e6	0.1244	–	–	–	0.1371	–	–	–

Electric Motor scenario (nondeterministic, unsolvable)								
	$h_{\max}$				h_{ff}			
	TT	PT	EN	PS	TT	PT	EN	PS
eu	0.1098	**859.972**	N/A	N/A	0.1168	–	N/A	N/A

Table 4
Evaluation metrics over the nondeterministic Chip Production scenario, both solvable (left) and unsolvable (right) variants, using the ltlfond2fond encoding and MyND with $h_{\max}$ and h_{ff} heuristics.

Chip Production scenario (nondeterministic)								
	$h_{\max}$				h_{ff}			
	TT	PT	EN	PS	TT	PT	EN	PS
cn1	0.1077	**0.023**	**32**	**17**	0.1227	**0.023**	**32**	**17**
cn2	0.128	**0.047**	202	**32**	0.1265	0.048	**193**	42
cn3	0.1197	**0.121**	1395	92	0.1296	0.148	**1100**	**57**
cn4	0.1295	0.521	7401	182	0.1259	**0.473**	**5474**	**127**
cn5	0.1375	4.456	31781	247	0.1357	**3.06**	**22389**	**192**
cn6	0.1432	64.081	114659	782	0.1214	**36.609**	**85027**	**147**
cn7	0.1469	–	–	–	0.1556	**881.816**	**345950**	**247**
cn8	0.1595	–	–	–	0.1681	–	–	–
cn9	0.1997	–	–	–	0.2002	–	–	–
cn10	0.2626	–	–	–	0.2496	–	–	–
cn11	0.3399	–	–	–	0.3278	–	–	–
cn12	0.5171	–	–	–	0.501	–	–	–

Chip Production scenario (nondeterministic, unsolvable)								
	$h_{\max}$				h_{ff}			
	TT	PT	EN	PS	TT	PT	EN	PS
cu1	0.1064	**0.019**	N/A	N/A	0.1259	0.02	N/A	N/A
cu2	0.1215	**0.031**	N/A	N/A	0.1167	0.033	N/A	N/A
cu3	0.1256	**0.063**	N/A	N/A	0.1211	0.097	N/A	N/A
cu4	0.1255	**0.168**	N/A	N/A	0.1267	0.17	N/A	N/A
cu5	0.1364	**0.371**	N/A	N/A	0.1364	0.421	N/A	N/A
cu6	0.1404	**1.01**	N/A	N/A	0.1422	1.144	N/A	N/A
cu7	0.1549	**3.657**	N/A	N/A	0.1505	3.751	N/A	N/A
cu8	0.1552	25.839	N/A	N/A	0.158	**24.798**	N/A	N/A
cu9	0.1806	289.501	N/A	N/A	0.183	**213.874**	N/A	N/A
cu10	0.2345	–	N/A	N/A	0.2484	–	N/A	N/A
cu11	0.3507	–	N/A	N/A	0.3339	–	N/A	N/A
cu12	0.5141	–	N/A	N/A	0.5054	–	N/A	N/A

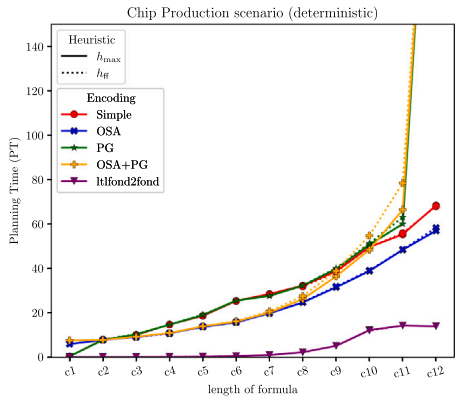


Fig. 9. Planning time (PT) metric for the Chip Production scenario (deterministic). Note that the scale for the y-axis is $[0s, 150s]$ for better comparison of the planning time measurements.

Chip Production scenario, the heuristic h_{ff} works slightly better than $h_{\max}$ on the PT metric and always better for the PS metric. In contrast, for the EN metric, there is no clear winner between the two alternatives. For the unsolvable variant of the scenario (Table 3, right), the execution with the h_{ff} heuristic did not finish within the timeout.

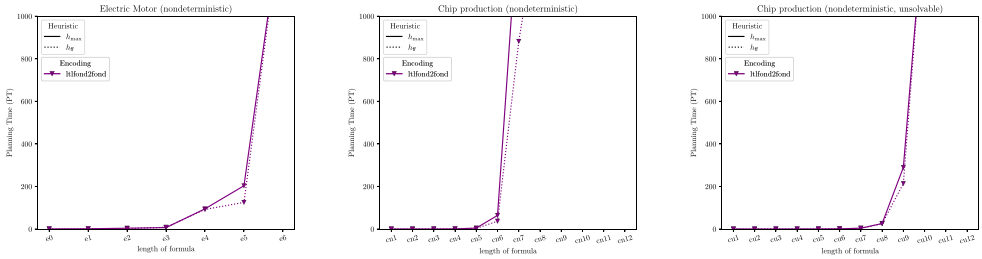
For better readability, we also show the plots for the PT metric for each scenario: Fig. 10(a) for the electric motor scenario, and Figs. 10(b) and 10(c) for the chip production scenario.

Overall, as expected, the performance was quite worse than the deterministic versions of the scenarios since nondeterministic planning problems are much more challenging to solve, both in theory and practice. It is perhaps a bit surprising that determining the unsolvability of certain variants is itself computationally challenging, even though the MyND planner manages to solve more instances in these unsolvable categories than in the solvable ones. To improve performance in these cases, an idea would be to develop pre-processing techniques that spot whether, for unavoidable process actions, there are only services whose execution leads to failure states. Other heuristics for the solvable case could instead avoid services with at least a failure state or prefer "safe" services (e.g., infallible services instead of breakable or irreparable ones). Another alternative is to explore better PDDL compilations, especially regarding using the `when` expressions (see discussion in Section 8.1).

9. 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. In order 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 [18,48,49].

This work is highly motivated by a renovated interest in service composition techniques with impactful applications in smart manufacturing [56–58]. 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) modeled as stochastic services. In future work, it will be fruitful to build on top



(a) PT for the Electric Motor scenario (nondeterministic). (b) PT for the Chip Production scenario (nondeterministic). (c) PT for the Chip Production scenario (unsolvable).

Fig. 10. Plots for the PT metric for each considered scenario.

of these foundations to 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.

CRediT authorship contribution statement

Giuseppe De Giacomo: Writing – original draft. **Marco Favorito:** Writing – original draft. **Luciana Silo:** Writing – original draft.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported in part by the ERC Advanced Grant WhiteMech (No. 834228), the PRIN project RIPER (No. 20203FFYLK), the PNRR MUR project FAIR (No. PE0000013), and the UKRI Erlangen AI Hub on Mathematical and Computational Foundations of AI. This work has been carried out while Luciana Silo was enrolled in the Italian National Doctorate on Artificial Intelligence run by Sapienza University of Rome.

Data availability

Data will be made available on request.

References

[1] D. Berardi, G. De Giacomo, M. Mecella, D. Calvanese, ComposingWeb services with nondeterministic behavior, in: 2006 IEEE International Conference on Web Services, ICWS'06, 2006, pp. 909–912, http://dx.doi.org/10.1109/ICWS.2006.45.
[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] R.I. Brafman, G. De Giacomo, M. Mecella, S. Sardina, Service composition in stochastic settings, in: AIxIA, 2017.
[5] G. De Giacomo, M. Mecella, F. Patrizi, Automated service composition based on behaviors: The Roman model, in: Web Services Foundations, Springer, 2014.
[6] H. Geffner, B. Bonet, A Concise Introduction to Models and Methods for Automated Planning, Morgan & Claypool Publishers, 2013.
[7] G. De Giacomo, F. Patrizi, S. Sardiña, Automatic behavior composition synthesis, Artificial Intelligence (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: Service-Oriented Computing, 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, W.M. Van der Aalst, A declarative approach for flexible business processes management, in: Business Process Management Workshops: BPM 2006 International Workshops, BPD, BPI, ENEI, GPWW, DPM, Semantics4ws, Vienna, Austria, September 4–7, 2006. Proceedings 4, Springer, 2006, pp. 169–180.
[12] M. Pešić, D. Bošnački, W.M. van der Aalst, Enacting declarative languages using LTL: avoiding errors and improving performance, in: Model Checking Software: 17th International SPIN Workshop, Enschede, the Netherlands, September 27–29, 2010. Proceedings 17, Springer, 2010, pp. 146–161.
[13] M. Pesic, H. Schonenberg, W.M. Van der Aalst, Declare: Full support for loosely-structured processes, in: EDOC, 2007.
[14] C. Di Ciccio, M. Montali, Declarative process specifications: Reasoning, discovery, monitoring, in: Process Mining Handbook, Springer, 2022.
[15] 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).
[16] G. De Giacomo, M.Y. Vardi, Linear temporal logic and linear dynamic logic on finite traces, in: IJCAI, 2013.
[17] G. De Giacomo, S. Rubin, Automata-theoretic foundations of FOND planning for LTL_f and LDL_f goals, in: IJCAI, 2018.
[18] A. Camacho, M. Bienvenu, S.A. McIlraith, Towards a unified view of AI planning and reactive synthesis, in: ICAPS, 2019.
[19] OMG, Case Management Model and Notation, version 1.0, 2014, URL https://www.omg.org/spec/CMMN/1.0/PDF.
[20] M.A. Marin, H. Lotriet, J.A. van der Poll, Metrics for the case management modeling and notation (CMMN) specification, in: SAICSIT, ACM, 2015, pp. 28:1–28:10.
[21] C. Di Ciccio, A. Marrella, A. Russo, Knowledge-intensive processes: Characteristics, requirements and analysis of contemporary approaches, J. Data Semant. 4 (1) (2015) 29–57.
[22] I. Routis, C. Bardaki, M. Nikolaidou, G. Dede, D. Anagnostopoulos, Exploring CMMN applicability to knowledge-intensive process modeling: An empirical evaluation by modelers, Knowl. Process. Manag. 30 (1) (2023) 33–54.
[23] M.A. Marin, H. Lotriet, J.A. van der Poll, Measuring method complexity of the case management modeling and notation (CMMN), in: SAICSIT, ACM, 2014, p. 209.
[24] J. Torres, J.A. Baier, Polynomial-time reformulations of LTL temporally extended goals into final-state goals, in: IJCAI, 2015.
[25] A. Camacho, S.A. McIlraith, Strong fully observable non-deterministic planning with LTL and LTL-f goals, in: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI, 2019, pp. 5523–5531.
[26] A. Cimatti, M. Pistore, M. Roveri, P. Traverso, Weak, strong, and strong cyclic planning via symbolic model checking, Artificial Intelligence (2003).
[27] A. Marrella, M. Mecella, S. Sardiña, Supporting adaptiveness of cyber-physical processes through action-based formalisms, AI Commun. (2018).
[28] E. Sirin, B. Parsia, D. Wu, J.A. Hendler, D.S. Nau, HTN planning for web service composition using SHOP2, J. Web Semant. (2004).
[29] J. Alves, J. Marchi, R. Fileto, M.A.R. Dantas, Resilient composition of web services through nondeterministic planning, in: ISCC, 2016.
[30] M. Pistore, A. Marconi, P. Bertoli, P. Traverso, Automated composition of web services by planning at the knowledge level, in: IJCAI, 2005.
[31] G.D. Giacomo, A.E. Gerevini, F. Patrizi, A. Saetti, S. Sardiña, Agent planning programs, Artificial Intelligence 231 (2016) 64–106.
[32] G. De Giacomo, P. Felli, F. Patrizi, S. Sardiña, Two-player game structures for generalized planning and agent composition, in: AAAI, 2010.
[33] M.Y. Vardi, Branching vs. Linear time: Final showdown, in: TACAS, in: Lecture Notes in Computer Science, vol. 2031, Springer, 2001, pp. 1–22.
[34] G. De Giacomo, M. Favorito, L. Silo, Composition of stochastic services for LTL_f goal specifications, in: FoIKS, 2024.

[35] G. De Giacomo, M.Y. Vardi, Linear temporal logic and linear dynamic logic on finite traces, in: IJCAI, IJCAI/AAAI, 2013, pp. 854–860.
[36] R.I. Brafman, G. De Giacomo, F. Patrizi, LTLf/LDLf non-Markovian rewards, in: AAAI, 2018.
[37] V. Fionda, G. Greco, LTL on finite and process traces: Complexity results and a practical reasoner, J. Artificial Intelligence Res. 63 (2018) 557–623.
[38] G. Röger, F. Pommerening, M. Helmert, Optimal Planning in the Presence of Conditional Effects: Extending LM-Cut with Context Splitting, in: ECAI, 2014.
[39] E. Scala, P. Haslum, S. Thiébaux, M. Ramírez, Subgoaling techniques for satisficing and optimal numeric planning, J. Artificial Intelligence Res. 68 (2020) 691–752.
[40] G. De Giacomo, M.Y. Vardi, Synthesis for LTL and LDL on finite traces, in: IJCAI, 2015.
[41] N. Yadav, S. Sardina, Decision theoretic behavior composition, in: AAMAS, 2011.
[42] F. Bacchus, F. Kabanza, Planning for temporally extended goals, Ann. Math. Artif. Intell. 22 (1–2) (1998) 5–27.
[43] F. Bacchus, F. Kabanza, Using temporal logics to express search control knowledge for planning, Artificial Intelligence (2000).
[44] 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.
[45] J.A. Baier, S.A. McIlraith, Planning with first-order temporally extended goals using heuristic search, in: AAAI, 2006.
[46] J.A. Baier, S.A. McIlraith, Planning with temporally extended goals using heuristic search, in: ICAPS, 2006.
[47] L. Bonassi, A.E. Gerevini, E. Scala, Planning with qualitative action-trajectory constraints in PDDL, in: IJCAI, 2022.
[48] 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.
[49] 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.
[50] G. De Giacomo, A. Di Stasio, F. Fuggitti, S. Rubin, Pure-past linear temporal and dynamic logic on finite traces, in: IJCAI, 2020.
[51] G. De Giacomo, M. Favorito, F. Leotta, M. Mecella, F. Monti, L. Silo, AIDA: A tool for resiliency in smart manufacturing, in: CAiSE, 2023.
[52] G. De Giacomo, M. Favorito, F. Leotta, M. Mecella, F. Monti, L. Silo, AIDA: A tool for resiliency in smart manufacturing, in: CAiSE Forum, in: Lecture Notes in Business Information Processing, vol. 477, Springer, 2023, pp. 112–120.
[53] M. Helmert, The fast downward planning system, J. Artificial Intelligence Res. 26 (2006) 191–246.
[54] R. Mattmüller, Informed progression search for fully observable nondeterministic planning=Informierte Vorwärtssuche für nichtdeterministisches Planen unter vollständiger Beobachtbarkeit (Ph.D. thesis), 2013.
[55] D. Tabakov, M.Y. Vardi, Experimental evaluation of classical automata constructions, in: LPAR, in: Lecture Notes in Computer Science, vol. 3835, Springer, 2005, pp. 396–411.
[56] 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.
[57] G. De Giacomo, P. Felli, B. Logan, F. Patrizi, S. Sardiña, Situation calculus for controller synthesis in manufacturing systems with first-order state representation, Artificial Intelligence (2022).
[58] G. De Giacomo, M.Y. Vardi, P. Felli, N. Alechina, B. Logan, Synthesis of orchestrations of transducers for manufacturing, in: AAAI, 2018.