DTU Library

Foundations of Reactive Synthesis for Declarative Process Specifications

Geatti, Luca; Montali, Marco; Rivkin, Andrey

Link back to DTU Orbit

Foundations of Reactive Synthesis for Declarative Process Specifications

Luca Geatti[1]**, Marco Montali**[2]**, Andrey Rivkin**[3]

[1]University of Udine
[2]Free University of Bozen-Bolzano
[3]Technical University of Denmark
luca.geatti@uniud.it, montali@inf.unibz.it, ariv@dtu.dk

Abstract

Given a specification of Linear-time Temporal Logic interpreted over finite traces (LTLf), the reactive synthesis problem asks to find a finitely-representable, terminating controller that reacts to the uncontrollable actions of an environment in order to enforce a desired system specification. In this paper we study, for the first time, the foundations of reactive synthesis for DECLARE, a well-established declarative, pattern-based business process modelling language grounded in LTLf. We provide a threefold contribution. First, we define a reactive synthesis problem for DECLARE. Second, we show how an arbitrary DECLARE specification can be polynomially encoded into an equivalent pure-past one in LTLf, and exploit this to define an EXPTIME algorithm for DECLARE synthesis. Third, we derive a symbolic version of this algorithm, by introducing a novel translation of pure-past temporal formulas into symbolic deterministic finite automata.

Introduction

Linear Temporal Logic (LTL) is one of the most widely studied modal logics for time, and is interpreted over infinite state sequences (or traces). Since its introduction (Pnueli 1977), LTL has been extensively employed in a variety of verification and synthesis tasks. While verification aims at checking the correctness of an LTL specification over a given dynamic system, synthesis uses an LTL specification to derive a corresponding correct-by-construction program (in the shape, *e.g.*, of a Mealy or Moore machine, I/O-transducer, or circuit) that realizes the specification. Extensive research has been conducted on different synthesis settings, considering in particular closed and *open* (also called *reactive*) systems, starting from the seminal works in (Harel and Pnueli 1984) and (Pnueli and Rosner 1989a). In the reactive setting, the system (referred to as Controller) interacts with its environment, which, in turn, can affect the behavior of Controller. Reactive synthesis is hence modeled as a two-player game between Controller, whose aim is to satisfy the formula, and Environment, who tries to violate it. The objective of the synthesis task is then to generate a program indicating which actions Controller should take to guarantee the satisfaction of the LTL specification of interest, no matter which actions

are taken by Environment. This problem was originally proposed in (Church 1962) and solved in (Buchi and Landweber 1990) for specifications written in Sequential Calculus. It was then shown to be 2EXPTIME-complete for LTL specifications (Rosner 1992). The high theoretical complexity and practical infeasibility of the original approach, based on the reduction to solving parity games on deterministic parity automata, inspired the scientific community to search for practically interesting fragments of LTL for which the synthesis problem is computationally more amenable.

In a variety of application domains, the dynamics of the system consist of unbounded, yet finite, traces (De Giacomo, De Masellis, and Montali 2014). This has led to the introduction of LTLf (De Giacomo and Vardi 2013) – LTL interpreted over *finite* traces. LTLf has been extensively studied within AI, formal methods, and Business Process Management (BPM) (Pesic and van der Aalst 2006; Fionda and Greco 2018; Maggi, Montali, and Peñaloza 2020; De Giacomo et al. 2022). In particular, extensive progress has been recently made in LTLf-based synthesis (Camacho et al. 2018; Zhu, Pu, and Vardi 2019; Bansal et al. 2020; Zhu et al. 2020; Giacomo et al. 2022), where the synthesised program always terminates (differently from the infinite-trace case). While the finite-trace semantics makes the problem more amenable to algorithmic optimization, it does not change the theoretical complexity of the synthesis problem: LTLf synthesis is 2EXPTIME-complete (De Giacomo and Vardi 2015). The classical LTLf synthesis algorithm requires to: (i) encode the formula into a nondeterministic automaton over finite words (NFA); (ii) turn the NFA into a deterministic automaton (DFA); (iii) solve a *reachability game*, in which Controller tries to force the game to reach a final state of the automaton. Each of the first two steps requires, in the worst case, an exponential amount of time in the size of its input.

In BPM, LTLf is employed to define the semantics of one of the most well-studied declarative process modelling languages, namely DECLARE (Pesic, Schonenberg, and van der Aalst 2007; Montali et al. 2010). DECLARE is a pattern-based language: constraints are defined based on a pre-defined set of unary or binary templates, formalized in LTLf, conjunctively related to each other. A flourishing line of research focuses on a variety of reasoning and analysis tasks for DECLARE, ranging from discovery of DECLARE

specifications from data representing executions of the process to offline and run-time verification of DECLARE specifications (Fionda and Greco 2018; Di Ciccio and Montali 2022; De Giacomo et al. 2022). Moreover, the importance of studying DECLARE goes beyond BPM. First, DECLARE is related to software engineering as its templates were originally derived from a catalogue of temporal properties distilled in an empirical software engineering study (Dwyer, Avrunin, and Corbett 1999). Second, as shown in (De Giacomo, De Masellis, and Montali 2014), such templates can be used in reasoning about actions as they are alike temporal patterns used in planning and, in particular, trajectory constraints (Bacchus and Kabanza 2000; Gerevini et al. 2009).

Reactive synthesis in this setting comes as a natural problem that, surprisingly, has never been studied in the literature. *In this paper, we formalize and study the reactive synthesis problem for declarative,* DECLARE *process specifications*, providing a threefold contribution.

First, we formalize the reactive synthesis problem of DECLARE, and use an example to illustrate its importance for the BPM and AI communities. Our reactive synthesis problem definition is driven by the following consideration. It is often neglected that processes are typically enacted by at least two parties: (i) the organization (and its internal resources) responsible for enacting the process, and (ii) external stakeholders taking uncontrollable, constrained decisions on how to progress with the execution. This calls for a form of *assume-guarantee* synthesis for DECLARE: given a declarative specification (*assumption*) regulating how the external activities can be executed, together with a declarative specification (*guarantee*) constraining executions of internal activities, the goal is to derive a program (*orchestration*) indicating how to ensure that the guarantee is respected under the hypothesis that the external stakeholders behave by respecting the assumption. We give a naïve, 2EXPTIME algorithm, that reduces the problem to LTLf synthesis.

Second, we show how to improve the naïve algorithm and obtain a *singly exponential-time algorithm for* DECLARE *synthesis*. This refined algorithm is based on the observation that, starting from pure past temporal formulas, it is possible to build language-equivalent deterministic finite automata of singly exponential size (Chandra, Kozen, and Stockmeyer 1981; De Giacomo et al. 2021). In particular, we introduce for the first time a *systematic encoding* of all DECLARE patterns into linear-size pure-past formulas of LTLf – a procedure that we call "pastification", following the terminology in (Cimatti et al. 2021)). As a by-product, this reveals the following fundamental properties, of independent theoretical interest: (i) the reactive synthesis problem of DECLARE is in EXPTIME; (ii) DECLARE is a fragment of LTLf with a polynomial pastification algorithm, a result that, as we show, cannot be achieved for full LTLf.

Third, we show a novel translation *from pure-past temporal formulas to symbolic, deterministic automata*, which leads to a purely symbolic version of the previous algorithm. This new version, which exploits the well-known practical benefits of the symbolic approach (Burch et al. 1992; McMillan 1993), opens the possibility of applying reactive synthesis of DECLARE in practice.

The rest of the paper is organized as follows. **??** reviews the necessary background. **??** defines the reactive synthesis problem for DECLARE based on the assume-guarantee paradigm, and shows a naïve approach for solving it. **??** presents the EXPTIME algorithm for DECLARE synthesis, based on the pastification of DECLARE formulas and **??** shows its symbolic version. Conclusions and future directions follow. Full proofs for lemmas and theorems can be found in (Geatti, Montali, and Rivkin 2022).

Background

Linear Temporal Logic over finite traces. Given a set Σ of proposition letters, a formula ϕ of LTLf is defined as follows (De Giacomo and Vardi 2013):

$$\phi := p \mid \neg p \mid \phi \vee \phi \mid \phi \wedge \phi \qquad \text{Boolean connectives}$$
$$\mid \mathsf{X}\phi \mid \widetilde{\mathsf{X}}\phi \mid \phi \mathbin{\mathsf{U}} \phi \mid \phi \mathbin{\mathsf{R}} \phi \qquad \text{future modalities}$$
$$\mid \mathsf{Y}\phi \mid \mathsf{Z}\phi \mid \phi \mathbin{\mathsf{S}} \phi \mid \phi \mathbin{\mathsf{T}} \phi \qquad \text{past modalities}$$

where $p \in \Sigma$. The future temporal operators X, $\widetilde{\mathsf{X}}$, U, and R are called *tomorrow*, *weak tomorrow*, *until*, and *release*, respectively. The past temporal operators Y, Z, S, and T are called *yesterday*, *weak yesterday*, *since*, and *triggers*, respectively. We use the standard shortcuts: $\top := p \vee \neg p$ and $\bot := p \wedge \neg p$ (for some $p \in \Sigma$); $\mathsf{F}\phi := \top \mathbin{\mathsf{U}} \phi$ (called *eventually*), $\mathsf{G}\phi := \bot \mathbin{\mathsf{R}} \phi$ (called *globally*), $\phi_1 \mathbin{\mathsf{W}} \phi_2 := \phi_1 \mathbin{\mathsf{U}} \phi_2 \vee \mathsf{G}\phi_1$ (called *weak until*), $\mathsf{O}\phi := \top \mathbin{\mathsf{S}} \phi$ (called *once*), and $\mathsf{H}\phi := \bot \mathbin{\mathsf{T}} \phi$ (called *historically*). Hereinafter, LTLf$_\mathsf{P}$ shall denote the *pure-past* fragment of LTLf (the fragment of LTLf without future temporal operators), and $|\phi|$ shall denote the *size* (the number of symbols) of any formula ϕ.

Formulas of LTLf over the alphabet Σ are interpreted over *finite traces* (or state sequences, or words), *i.e.*, sequences $(2^\Sigma)^+$. We call *general finite trace semantics* the interpretation under such structures. Let $\sigma = \langle \sigma_0, \ldots, \sigma_{n-1} \rangle \in (2^\Sigma)^+$. Let $|\sigma| = n$ be the *length* of σ. With $\sigma_{[i,j]}$ (for some $0 \le i \le j < |\sigma|$) we denote the subinterval $\langle \sigma_i, \ldots, \sigma_j \rangle$ of σ. The *satisfaction* of an LTLf formula ϕ by σ at time $0 \le i < |\sigma|$, denoted by $\sigma, i \models \phi$, is defined as follows:

- $\sigma, i \models p$ iff $p \in \sigma_i$; $\sigma, i \models \neg p$ iff $p \notin \sigma_i$;
- $\sigma, i \models \phi_1 \vee \phi_2$ iff $\sigma, i \models \phi_1$ or $\sigma, i \models \phi_2$;
- $\sigma, i \models \phi_1 \wedge \phi_2$ iff $\sigma, i \models \phi_1$ and $\sigma, i \models \phi_2$;
- $\sigma, i \models \mathsf{X}\phi$ iff $i + 1 < |\sigma|$ and $\sigma, i + 1 \models \phi$;
- $\sigma, i \models \widetilde{\mathsf{X}}\phi$ iff either $i + 1 = |\sigma|$ or $\sigma, i + 1 \models \phi$;
- $\sigma, i \models \mathsf{Y}\phi$ iff $i > 0$ and $\sigma, i - 1 \models \phi$;
- $\sigma, i \models \mathsf{Z}\phi$ iff either $i = 0$ or $\sigma, i - 1 \models \phi$;
- $\sigma, i \models \phi_1 \mathbin{\mathsf{U}} \phi_2$ iff there exists $i \le j < |\sigma|$ s.t. $\sigma, j \models \phi_2$, and $\sigma, k \models \phi_1$ for all k, with $i \le k < j$;
- $\sigma, i \models \phi_1 \mathbin{\mathsf{S}} \phi_2$ iff there exists $j \le i$ s.t. $\sigma, j \models \phi_2$, and $\sigma, k \models \phi_1$ for all k, with $j < k \le i$;
- $\sigma, i \models \phi_1 \mathbin{\mathsf{R}} \phi_2$ iff either $\sigma, j \models \phi_2$ for all $i \le j < |\sigma|$, or there exists $i \le k < |\sigma|$ s.t. $\sigma, k \models \phi_1$ and $\sigma, j \models \phi_2$ for all $i \le j \le k$;
- $\sigma, i \models \phi_1 \mathbin{\mathsf{T}} \phi_2$ iff either $\sigma, j \models \phi_2$ for all $0 \le j \le i$, or there exists $k \le i$ s.t. $\sigma, k \models \phi_1$ and $\sigma, j \models \phi_2$ for all $i \ge j \ge k$.

We say that σ is a *model* of ϕ (written as $\sigma \models \phi$) iff $\sigma, 0 \models \phi$. The *language* (of finite words) of ϕ, denoted by $\mathcal{L}(\phi)$, is the set of traces $\sigma \in (2^\Sigma)^+$ s.t. $\sigma \models \phi$. We say that two formulas

$\phi, \psi \in$ LTLf are *equivalent* iff $\mathcal{L}(\phi) = \mathcal{L}(\psi)$. If ϕ belongs to LTLf$_P$ (*i.e.*, pure past fragment of LTLf), then we interpret ϕ at the last time point of the trace, *i.e.*, we say that $\sigma \in (2^\Sigma)^+$ is a model of ϕ if and only if $\sigma, |\sigma| - 1 \models \phi$.

The DECLARE **Language.** DECLARE (Pesic, Schonenberg, and van der Aalst 2007; Montali et al. 2010) is a language for the declarative specification of flexible processes. We refer to (Montali 2010; Reichert and Weber 2012) for a thorough treatment of declarative processes and flexibility in process management. A DECLARE specification consists of a set of *temporal constraints* used to implicitly regulate which traces conform to the process and which do not: a trace is conforming if it satisfies all the constraints captured by the specification. Constraints are defined by instantiating pre-defined templates on a set of *actions*, *i.e.*, atomic tasks representing units of work in the process. This can be extended to boolean combinations of actions (and results of this paper seamlessly cover such boolean combinations), but for the sake of simplicity, we keep single actions. Every template comes also with a graphical notation, supporting the visual presentation of DECLARE specifications (Pesic, Schonenberg, and van der Aalst 2007; Montali et al. 2010).

DECLARE assumes that exactly one action can be executed at each time point, and that each execution eventually terminates. Thus, the semantics of DECLARE (De Giacomo, De Masellis, and Montali 2014) is given by means of LTLf interpreted over *simple finite traces* $\sigma = \langle \sigma_0, \ldots, \sigma_n \rangle$ in Σ^+ s.t. $|\sigma_i| = 1$ for any $0 \leq i < |\sigma|$. We indicate this interpretation as *simple finite trace semantics*. This can be recast in the general finite trace semantics by adding the following LTLf formula (Montali 2010; Fionda and Greco 2018): $\texttt{simple}(\Sigma) := G(\bigvee_{p \in \Sigma} p \land \bigwedge_{p \neq q \in \Sigma} \neg(p \land q))$.

Table 1 shows the catalog of DECLARE templates, indicating template names (with p and q as placeholders for actions) and their corresponding LTLf formalization under the simple trace semantics. From now on, with some abuse of notation, with *"DECLARE pattern"* we refer to its corresponding formalization in LTLf, and with *"DECLARE specification"* to a conjunction of DECLARE patterns.

Finite-State Automata. Since LTLf is interpreted over finite traces, its automata-theoretic characterization is grounded in automata over *finite words* (De Giacomo and Vardi 2013). Given an alphabet Σ, $\sigma \in \Sigma^*$ is a *finite word* (or simply a *word*). We use two representations of automata, the (classical) explicit-state one, and a symbolic one.

Definition 1. *A nondeterministic finite automaton (*NFA*) $\mathcal{A}$ is a tuple $(\Sigma, Q, I, \delta, F)$ s.t.: (i) Σ is a finite alphabet; (ii) Q is a set of states; (iii) $I \subseteq Q$ is the set of initial states; (iv) $\delta : Q \times \Sigma \to 2^Q$ is the transition relation; (v) $F \subseteq Q$ is the set of final states. A deterministic finite automaton (*DFA*) is an* NFA *such that $|I| = 1$ and $\delta : Q \times \Sigma \to Q$.*

Given an NFA $\mathcal{A}$ with set of states Q, with $|\mathcal{A}|$ we denote $|Q|$. Given an NFA $\mathcal{A} = (\Sigma, Q, I, \delta, F)$ and a word $\sigma = \langle \sigma_0, \ldots, \sigma_n \rangle \in \Sigma^*$, a *run π induced by σ in $\mathcal{A}$* is a (finite) sequence of states $\langle q_0, \ldots, q_{n+1} \rangle \in Q^*$ s.t. $q_0 \in I$ and $q_{i+1} \in \delta(q_i, \sigma_i)$, for any $i \geq 0$. A run $\pi = \langle q_0, \ldots, q_{n+1} \rangle$ is *accepting* iff $q_{n+1} \in F$. The language of $\mathcal{A}$, denoted as $\mathcal{L}(\mathcal{A})$, is the set of words σ s.t. there exists at least one accepting run induced by σ in $\mathcal{A}$. Each NFA can be turned into a language-equivalent DFA of exponential size in the worst-case (see, e.g., (Hopcroft, Motwani, and Ullman 2001)). Each language expressible in LTLf can be recognized by an NFA, whose size is in the worst case exponential in the size of the formula (De Giacomo and Vardi 2013). Even though LTLf$_P$ has the same expressive power as LTLf (Lichtenstein, Pnueli, and Zuck 1985), LTLf$_P$ admits a construction of equivalent DFAs of *only* singly exponential size (since "the past already happened", pure-past formulas can be turned into automata without introducing nondeterminism) (De Giacomo et al. 2021; Cimatti et al. 2021). .

Proposition 1. *For any* LTLf$_P$ *formula ϕ with $n = |\phi|$, there exists a* DFA $\mathcal{A}$ *s.t. $\mathcal{L}(\phi) = \mathcal{L}(\mathcal{A})$ and $|\mathcal{A}| \in 2^{O(n)}$.*

The drawback of the explicit-state representation of automata is the required memory, which can be prohibitively large for analysis tasks such as model checking and reactive synthesis. The symbolic representation aims at overcoming this issue by representing automata through Boolean formulas (McMillan 1993). In the average case, this representation can be *exponentially more succinct* than the explicit one.

Definition 2. *A symbolic NFA over the alphabet Σ is a tuple $\mathcal{S} = (X \cup \Sigma, I(X), T(X, \Sigma, X'), F(X, \Sigma))$, where (i) X is a set of* state variables, *(ii) $I(X)$ and $T(X, \Sigma, X')$, with $X' = \{x' \mid x \in X\}$, are Boolean formulas that respectively define the set of initial states and the transition relation, and (iii) $F(X, \Sigma)$ is a Boolean formula over $X \cup \Sigma$ that defines the set of final states.*

For a symbolic NFA $\mathcal{S} = (X \cup \Sigma, I(X), T(X, \Sigma, X'), F(X, \Sigma))$, $|\mathcal{S}|$ denotes the sum of the number of symbols in I, T and F. Symbolic NFAs can also be refined into symbolic DFAs. A symbolic DFA is a symbolic NFA such that: (i) its formula $I(X)$ has exactly one satisfying assignment; (ii) its transition relation is of the form $T(X, \Sigma, X') := \bigwedge_{x \in X}(x' \leftrightarrow \beta_x(V))$, where $\beta_x(V)$ is a Boolean formula with $V = X \cup \Sigma$, for each $x \in X$.

Given a symbolic NFA $\mathcal{A} = (X \cup \Sigma, I(X), T(X, \Sigma, X'), F(X, \Sigma))$, a run $\tau = \langle \tau_0, \ldots, \tau_n \rangle$ (for $n \in \mathbb{N}$) is a finite sequence of pairs $\tau_i := (X_i, \Sigma_i) \subseteq 2^X \times 2^\Sigma$ (representing the state and input variables that are supposed to hold at time point i) that satisfies the following two conditions: (i) $\tau_0 \models I(X)$; (ii) $\tau_i, \tau_{i+1} \models T(X, \Sigma, X')$, for each $0 \leq i \leq n$, when τ_i is used for interpreting the variables in X and Σ, and τ_{i+1} is used for interpreting the variables in X'. A run $\tau = \langle (X_0, \Sigma_0), \ldots, (X_n, \Sigma_n) \rangle$ is induced by the word $\langle \sigma_0, \ldots, \sigma_n \rangle \in \Sigma^*$ (for $n \in \mathbb{N}$) iff $\sigma_i = \Sigma_i$ for all $0 \leq i \leq n$. A run τ is *accepting* iff $\tau_n \models F(X, \Sigma)$. A word σ is *accepted* by $\mathcal{A}$ iff there exists an accepting run induced by σ in $\mathcal{A}$. The *language* of $\mathcal{A}$, denoted by $\mathcal{L}(\mathcal{A})$, is the set of all and only the infinite words accepted by $\mathcal{A}$.

Reactive Synthesis of DECLARE

We now recap the realizability and reactive synthesis problems of LTLf, and then define their DECLARE counterpart. We then illustrate the main ideas on an example and show how such problems can be solved using a naïve approach.

Realizability and Synthesis of LTLf

Realizability (Pnueli and Rosner 1989b) aims at establishing whether, given an LTLf formula ϕ over two sets $\mathcal{U}$ and $\mathcal{C}$ of, respectively, *uncontrollable* and *controllable* variables, there exists a strategy which, no matter how the variables in $\mathcal{U}$ are set, chooses the value of the variables in $\mathcal{C}$ so that ϕ is satisfied. *Reactive synthesis* is the problem of synthesizing such a strategy (e.g., in the form of a Mealy or Moore machine (Vardi 1995)) in case the formula is realizable. Hereinafter, we fix Σ as the union of $\mathcal{C}$ and $\mathcal{U}$, with $\mathcal{C} \cap \mathcal{U} = \varnothing$.

Definition 3. *A strategy for Controller is a function $s : (2^{\mathcal{U}})^+ \to 2^{\mathcal{C}}$ that, for any finite sequence $\mathsf{U} = \langle \mathsf{U}_0, \ldots, \mathsf{U}_n \rangle$ of choices by Environment, determines the choice $\mathsf{C}_n = s(\mathsf{U})$ of Controller.*

For an infinite sequence of choices by Environment $\mathsf{U} = \langle \mathsf{U}_0, \mathsf{U}_1, \ldots \rangle \in (2^{\mathcal{U}})^\omega$, we denote by $\mathtt{res}(s, \mathsf{U}) = \langle \mathsf{U}_0 \cup s(\langle \mathsf{U}_0 \rangle), \mathsf{U}_1 \cup s(\langle \mathsf{U}_0, \mathsf{U}_1 \rangle), \ldots \rangle$ the trace resulting from reacting to U according to s. Realizability is usually modeled as a two-player game between Environment and Controller, who try to respectively violate and fulfill the given formula.

Definition 4. *Let ϕ be an LTLf formula over Σ. We say that ϕ is realizable (over finite words) iff there exists a strategy $s : (2^{\mathcal{U}})^+ \to 2^{\mathcal{C}}$ s.t., for any infinite sequence $\mathsf{U} = \langle \mathsf{U}_0, \mathsf{U}_1, \ldots \rangle$ in $(2^{\mathcal{U}})^\omega$ there exists $k \in \mathbb{N}$ for which $\mathtt{res}(s, \mathsf{U})_{[0,k]} \models \phi$. If ϕ is realizable, reactive synthesis is the problem of computing such a strategy s.*

The baseline algorithm for solving realizability of an LTLf formula ϕ works as follows (De Giacomo and Vardi 2015): (i) build an NFA for ϕ; (ii) transform the NFA into a language-equivalent DFA $\mathcal{A}$; (iii) play a *reachability game* over $\mathcal{A}$ to establish whether Controller can force the game to reach a final state of the automaton. If this is the case, then ϕ is realizable, otherwise it is unrealizable. The realizability problem for LTLf is 2EXPTIME-complete.

Realizability over Simple Traces

We now use the standard realizability problem for LTLf, defined over general finite traces as per Definition 4, as a starting point towards realizability for DECLARE. As an intermediate step, we need to recast Definition 4 by considering now simple finite traces only, instead of general ones[1]. This brings *two* significant differences. First, in the case of simple traces, we impose that both players can play only one proposition letter from their set. Second, we impose a strict alternation: Environment starts to play at the first time point and whenever Environment plays at time point i, Controller plays at time point $i + 1$, unless the play has finished before.

We begin by defining the basic building blocks. A *simple strategy* for Controller is a function $s : (\mathcal{U})^+ \to \mathcal{C}$ that, for every finite sequence of elements in $\mathcal{U}$, determines an element in the set $\mathcal{C}$. Let $s : (\mathcal{U})^+ \to \mathcal{C}$ be a simple strategy and let $\mathsf{U} = \langle \mathsf{U}_0, \mathsf{U}_1, \ldots \rangle \in (\mathcal{U})^\omega$ be an infinite *simple trace* of choices by Environment. We denote by $\mathtt{simres}(s, \mathsf{U})$ the state sequence $\langle \mathsf{U}_0, s(\langle \mathsf{U}_0 \rangle), \mathsf{U}_1, s(\langle \mathsf{U}_0, \mathsf{U}_1 \rangle), \ldots \rangle$ resulting

---

[1]This is, to the best of our knowledge, the first time that the realizability problem over *simple finite traces* is defined.

from the alternation between the choices of Environment and the corresponding choices of $s(\cdot)$. Crucially, for any $k \geq 0$, $\mathtt{simres}(s, \mathsf{U})_{[0,k]}$ is a *simple* finite trace. We then define realizability over simple finite traces as follows.

Definition 5. *Let ϕ be an LTLf formula over the alphabet Σ. The formula ϕ is realizable over simple finite traces iff there exists a simple strategy $s : (\mathcal{U})^+ \to \mathcal{C}$ such that, for any infinite sequence $\mathsf{U} = \langle \mathsf{U}_0, \mathsf{U}_1, \ldots \rangle$ in $(\mathcal{U})^\omega$, there exists $k \in \mathbb{N}$ such that $\mathtt{simres}(s, \mathsf{U})_{[0,k]} \models \phi$.*

Strictly alternating game rounds appear as the most natural choice when dealing with *simple finite* traces: *(i)* the two players cannot play at the same round (this yields non-simple traces); *(ii)* allowing players to play for rounds of unbounded length introduces additional challenges related to fairness (Zhu et al. 2020). Also, rounds of any bounded length can be simulated with Definition 5 by adding two auxiliary variables that model a `no-op` action of the players and LTLf constraints that, whenever the round belongs to one of the players, force the other to choose the `no-op` action. Finally, even in the strict alternating case of Definition 5, introducing `no-op` actions is conceptually useful as it allows a player to release control to the other player.

Realizability and Synthesis of DECLARE

We now turn to realizability of DECLARE. We start from the key consideration that in the case of agents in AI (Zhu and De Giacomo 2022), as well as in BPM, there is background knowledge on how Environment operates. In particular, in BPM the external stakeholders participate to the process in a constrained way: when it is their turn, they take (arbitrary) decisions on which next action to trigger but only on the subset of all actions made available by the information system supporting the enactment of the process (Dumas et al. 2018).

DECLARE hence calls for an *assume-guarantee* paradigm to synthesis: Controller guarantees to enforce certain DECLARE constraints, under the assumption that Environment satisfies its own set of DECLARE constraints. More specifically, the realizability problem takes as input two DECLARE specifications, one for the assumption and the other for the guarantee, and aims at finding a strategy for Controller fulfilling all the guarantee constraints, regardless on how Environment behaves in the space of possibilities given by the assumption constraints. Environment loses the realizability game if it violates its assumptions.

Definition 6. *Given two DECLARE specifications ϕ_E (Environment) and ϕ_C (Controller), the realizability problem of (ϕ_E, ϕ_C) is the problem of establishing whether $\phi_E \to \phi_C$ is realizable over simple finite traces. Whenever it is so, reactive synthesis is the problem of computing a simple strategy.*

Example 2. *This assume-guarantee approach to DECLARE synthesis is also significant as DECLARE employs predefined LTLf patterns, composed conjunctively. By considering the actual DECLARE templates (cf. Table 1), one can easily notice that a direct adaptation of LTLf synthesis to DECLARE would not be a viable approach. This would indeed result in a single monolithic DECLARE specification with actions partitioned into Environment and Controller*
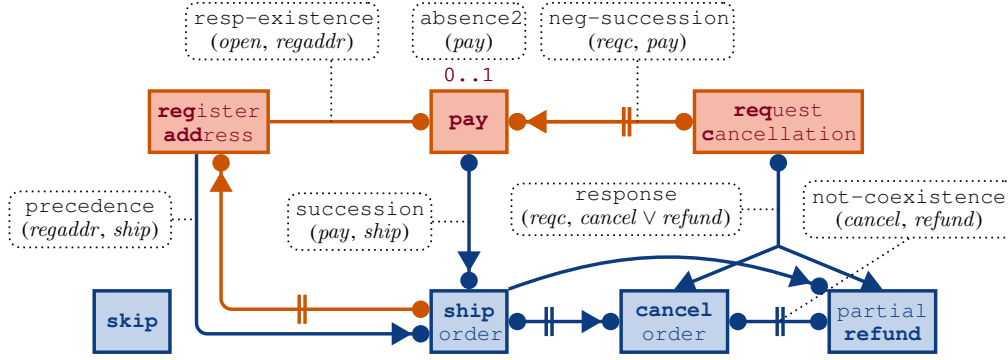
Figure 1: An order handling process in DECLARE, following the DECLARE graphical notation. Rectangles denote actions, connectors denote constraints. Customer actions and constraints are depicted in orange, those of the seller in blue.

actions, with constraints quickly leading to unrealizability. For example, constraints solely involving Environment activities are unrealizable, unless additional assumptions are posed on Environment.

The following example shows the benefits of the assume-guarantee paradigm for reactive synthesis of DECLARE.

Consider a seller that needs to orchestrate an order-to-delivery DECLARE *process, interacting with external customers interested in ordering material from the shop. The customer plays the role of Environment, and the seller that of Controller. A fragment of this process is shown in Fig. 1. The process actions are partitioned into those of the seller (e.g.,* ship order*) and of the customer (e.g.,* pay*). Both the* Environment assumption *(in orange) and* Controller guarantee *(in blue) contain constraints referring solely to actions of the customer or seller as well as constraints linking actions of both parties.*

*The specification does not cover the order creation and selection of items by the customer, which is assumed to be already completed when the process is enacted. The process then executes as follows. The customer can register their address (regaddr), pay for the created order (pay), or request its cancellation (reqCancel). Whenever an order is paid, the customer has to eventually register their address, unless this has been already done (*resp-existence(open, regaddr)*); further address registrations can be performed to update the address data. The payment occurs in a single installment (*absence2(pay)*), and can be performed only if the customer has not previously issued a request for order cancellation (*neg-succession(reqc, pay)*).*

*The seller can ship the order and handle full cancellation (cancel) or partial refund (refund). It also has a no-op action (skip) to return control to the customer. The seller's portion of the process is regulated by the following constraints. Full and partial cancellations are mutually exclusive ((*not-coexistence(cancel, refund)*). Full cancellation can be chosen only if the order has not been shipped (*neg-succession(ship, cancel)*), while partial refund only for shipped orders (*precedence(cancel, refund)*).*

Finally, there are constraints mutually relating the actions of the two parties, and that must be assigned to the assumption or the guarantee depending on their contrac-

*tual nature. For example, as part of the Environment assumption, the customer agrees that they will not change the delivery address anymore once the order is shipped (*neg-succession(ship, regaddr)*).*

Obverse that the seller can indeed synthesize a strategy to orchestrate the process. It works as follows. If the customer pays, the seller needs to wait for the registration of the address unless this was already done. Once both activities are executed, the seller ships. Upon a cancellation request, the shop reacts as follows: if the order has not been shipped yet, it is canceled, otherwise it is partially refunded.

To solve realizability and reactive synthesis from DECLARE specifications, we resort to a reduction to realizability and synthesis of LTLf over general finite traces. To do so, we first need to introduce two LTLf formulas $\texttt{simple}_{\texttt{Env}}(\mathcal{U})$ and $\texttt{simple}_{\texttt{Con}}(\mathcal{C}))$ to enforce that the game of the play is a simple trace and to force, respectively, Environment to play at even time points, and Controller to play at odd ones. Technically:

$$\texttt{simple}_{\texttt{Env}}(\mathcal{U}) \coloneqq \bigvee_{u \in \mathcal{U}} u \wedge \mathsf{G}\big(\bigvee_{u \in \mathcal{U}} u \rightarrow \big(\bigwedge_{\substack{u \neq u' \\ \in \mathcal{U}}} \neg(u \wedge u') \wedge$$

$$\widetilde{\mathsf{X}}\big(\bigwedge_{u \in \mathcal{U}} \neg u \wedge \widetilde{\mathsf{X}} \bigvee_{u \in \mathcal{U}} u\big)\big)\big)$$

$$\texttt{simple}_{\texttt{Con}}(\mathcal{C}) \coloneqq \bigwedge_{c \in \mathcal{C}} \neg c \wedge \mathsf{G}\big(\bigwedge_{c \in \mathcal{C}} \neg c \rightarrow \widetilde{\mathsf{X}}\big(\bigvee_{c \in \mathcal{C}} c \wedge$$

$$\bigwedge_{\substack{c \neq c' \\ \in \mathcal{C}}} \neg(c \wedge c') \wedge \widetilde{\mathsf{X}} \bigwedge_{c \in \mathcal{C}} \neg c\big)\big)$$

Note the role of the *weak tomorrow* operators in $\texttt{simple}_{\texttt{Env}}(\mathcal{U})$ and $\texttt{simple}_{\texttt{Con}}(\mathcal{C})$, which ensure that the game can stop at any moment. By carefully employing these two formulas, we obtain the following reduction.

Lemma 3. *Let ϕ_E and ϕ_C be two* DECLARE *specifications over the set of actions Σ. It holds that (ϕ_E, ϕ_C) is realizable in the sense of Definition 6 iff the* LTLf *formula $\texttt{simple}_{\texttt{Con}}(\mathcal{C}) \wedge ((\texttt{simple}_{\texttt{Env}}(\mathcal{U}) \wedge \phi_E) \rightarrow \phi_C)$ is realizable in the sense of Definition 4.*

We can then tackle DECLARE realizability by constructing the LTLf formula from Lemma 3, then feeding it into

classical algorithms for LTLf realizability (De Giacomo and Vardi 2015). The same applies to reactive synthesis as well. Such algorithms, however, have a 2EXPTIME complexity.

Applications of reactive synthesis for DECLARE. As it has been discussed in (Geatti, Montali, and Rivkin 2023), realizability (and synthesis tasks) are crucial for refined (declarative) process specifications consisting of two parts: one internal (orchestrator/Controller) and one external (external participant/environment). While the interface between such parts is implicitly embedded into the binary DECLARE patterns including actions of both Controller and Environment, there is no guarantee that such collaborative specification is actually executable. This is where realizability can be used to check the implementability of the specification (that is, verify whether the controller can execute the process guaranteeing the constraints from the specification, provided that Environment satisfies its assumptions) and eventually obtain an orchestration mechanism – a strategy witnessing the implementability. Such a strategy provides a specific behaviour for Controller ensuring that, whenever the environment behaves in accordance to the assumptions in the specification, the resulting reactions yield a simple process trace that satisfies the guarantees.

In practice, reactive synthesis comes in handy when collaborating stakeholders (executing a single-organization process and its external clients, or an inter-organizational process) prefer to specify interfaces of their processes instead of exposing the entire process models and trying to achieve explicit solutions on the cost of multiple and time-consuming process integration sessions. Like that, the feasibility of the proposed interface together with its implementation strategy can be outsourced to the reactive synthesis framework discussed in this paper.

Efficient Reactive Synthesis for DECLARE

In this section, we give a more efficient algorithm for the realizability problem of DECLARE that works in singly exponential time. It is based on two steps: (i) a (linear) encoding of any DECLARE specification into an equivalent LTLf$_P$ formula; (ii) construction of a singly-exponential DFA (recall Proposition 1) to be used as a reachability game arena.

Pastification for DECLARE

The first step of our algorithm is called *pastification* – transformation of a DECLARE specification into an equivalent one that uses only past temporal operators. W.l.o.g., in this section, we assume general finite trace semantics.

Definition 7. *Let ϕ be a* DECLARE *specification over the alphabet Σ. A* pastification *of ϕ, denoted as* $\mathrm{past}(\phi)$*, is a formula of* LTLf$_P$ *over Σ s.t., for any finite trace $\sigma \in (2^\Sigma)^+$, it holds that $\sigma, 0 \models \phi \Leftrightarrow \sigma, |\sigma| - 1 \models \mathrm{past}(\phi)$*

We show that every DECLARE pattern can be pastified into a formula with size linear in the size of the input.

Theorem 4. *For each pattern in Table 1, let ϕ (resp., ψ) be the formula in the second (resp., third) column. It holds that $\psi = \mathrm{past}(\phi)$ and $|\psi| \in O(|\phi|)$.*

Algorithm 1 *synth*-DECLARE

Input: $\Sigma = \mathcal{C} \cup \mathcal{U}$, environment specification ϕ_E, controller specification ϕ_C
1: $\psi_{\mathtt{SimpleCon}} \longleftarrow \mathrm{past}(\mathtt{simple}_{\mathtt{Con}}(\mathcal{C}))$
2: $\psi_{\mathtt{SimpleEnv}} \longleftarrow \mathrm{past}(\mathtt{simple}_{\mathtt{Env}}(\mathcal{U}))$
3: $\psi_E \longleftarrow \mathrm{past}(\phi_E)$
4: $\psi_C \longleftarrow \mathrm{past}(\phi_C)$
5: $\mathcal{A} \longleftarrow \mathrm{build\text{-}DFA}(\psi_{\mathtt{SimpleCon}} \wedge ((\psi_{\mathtt{SimpleEnv}} \wedge \psi_E) \rightarrow \psi_C))$
6: $s \longleftarrow \mathrm{realize\text{-}and\text{-}extract}(\mathcal{A})$
7: **if** s has been found **then**
8:     **return** s;
9: **else**
10:     **return** unrealizable.
11: **end if**

The same theorem directly generalizes to full DECLARE specifications, *i.e.*, to any conjunction of patterns.

It is worth pointing out the following property of Definition 7: since the formula $\widetilde{X}(\bot)$ is true iff it is interpreted at the last time point of a (finite) trace, Definition 7 amounts to require that the formula $\phi \leftrightarrow F(\widetilde{X}(\bot) \wedge \mathrm{past}(\phi))$ is *valid*. We also checked the correctness of the pastification of each DECLARE pattern ψ by checking the validity of formulas of the previous type with the BLACK tool (Geatti et al. 2021; Geatti, Gigante, and Montanari 2021), by reducing the validity checking of ψ to the satisfiability checking of $\neg\psi$.

To check realizability, we require the two formulas from Lemma 3. We show that they are also linear-size pastifiable.

Lemma 5. *There exist two linear-size pastifications of* $\mathtt{simple}_{\mathtt{Env}}(\mathcal{U})$ *and* $\mathtt{simple}_{\mathtt{Con}}(\mathcal{C})$.

We remark that the property of admitting a polynomial-size pastification is very rare. As a matter of fact, it is impossible for full LTLf to have such a feature, because realizability of LTLf$_P$ is an EXPTIME-complete problem (Artale et al. 2023b), while realizability of LTLf is 2EXPTIME-complete. While it is known that LTLf with only X as temporal operator admits a polynomial-size pastification (Maler, Nickovic, and Pnueli 2005), considering also the F operator makes the pastification exponential (Artale et al. 2023a).

A Singly Exponential-Time Algorithm

We rely on **??** to obtain the next central result.

Theorem 6. DECLARE *realizability is in* EXPTIME.

The proof is given by Algorithm 1, which shows an EXPTIME procedure for DECLARE realizability. Starting from (ϕ_E, ϕ_C), the algorithm applies pastification to all subformulas of $\phi = \mathtt{simple}_{\mathtt{Con}}(\mathcal{C}) \wedge ((\mathtt{simple}_{\mathtt{Env}}(\mathcal{U}) \wedge \phi_E) \rightarrow \phi_C)$, obtaining an equivalent pure-past formula $\mathrm{past}(\phi)$. By Theorem 4 and Lemma 5, $|\mathrm{past}(\phi)|$ is linear in $|\phi_E|$ and $|\phi_C|$. Procedure build-DFA then builds a DFA $\mathcal{A}$ that has a singly exponential size with respect to $|\phi|$, and that recognizes the language of such pure-past formula (see (De Giacomo et al. 2021) for details on this procedure). Finally, the realize-and-extract procedure solves a reachability game (De Alfaro, Henzinger, and Kupferman 2007) on $\mathcal{A}$ to determine whether Controller can force reachability of a final state in the automaton (see (Jacobs et al. 2017) for

Pattern	LTLf formalization	Pastification $(\mathrm{past}(\cdot))$
existence(p)	$\mathsf{F}(p)$	$\mathsf{O}(p)$
absence2(p)	$\neg\mathsf{F}(p \wedge \mathsf{XF}(p))$	$\mathsf{H}(p \to \mathsf{ZH}(\neg p))$
choice(p,q)	$\mathsf{F}(p) \vee \mathsf{F}(q)$	$\mathsf{O}(p \vee q)$
exc-choice(p,q)	$(\mathsf{F}(p) \vee \mathsf{F}(q)) \wedge$ $\neg(\mathsf{F}(p) \wedge \mathsf{F}(q))$	$\mathsf{O}(p \vee q) \wedge$ $(\mathsf{H}(\neg p) \vee \mathsf{H}(\neg q))$
resp-existence(p,q)	$\mathsf{F}(p) \to \mathsf{F}(q)$	$\mathsf{H}(\neg p) \vee \mathsf{O}(q)$
coexistence(p,q)	$\mathsf{F}(p) \leftrightarrow \mathsf{F}(q)$	$(\mathsf{H}(\neg p) \vee \mathsf{O}(q)) \wedge$ $(\mathsf{H}(\neg q) \vee \mathsf{O}(p))$
response(p,q)	$\mathsf{G}(p \to \mathsf{F}(q))$	$q \mathrel{\mathsf{T}} (\neg p \vee q)$
precedence(p,q)	$(\neg q) \mathrel{\mathsf{W}} (p)$	$\mathsf{H}(q \to \mathsf{O}(p))$
succession(p,q)	$\mathsf{G}(p \to \mathsf{F}(q)) \wedge$ $(\neg q) \mathrel{\mathsf{W}} (p)$	$p \mathrel{\mathsf{T}} (\neg p \vee q) \wedge$ $\mathsf{H}(q \to \mathsf{O}(p))$
alt-response(p,q)	$\mathsf{G}(p \to \mathsf{X}((\neg p) \mathrel{\mathsf{U}} q))$	$(p \vee q) \mathrel{\mathsf{T}} (\neg p) \wedge \mathsf{H}(q \to \mathsf{Z}(q \mathrel{\mathsf{T}} ((p \wedge$ $\neg q) \to \mathsf{Z}(q \mathrel{\mathsf{T}} \neg p))))$
alt-precedence(p,q)	$((\neg q) \mathrel{\mathsf{W}} p) \wedge$ $\mathsf{G}(q \to \widetilde{\mathsf{X}}((\neg q) \mathrel{\mathsf{W}} p))$	$\mathsf{H}(q \to \mathsf{O}(p)) \wedge$ $\mathsf{H}((q \wedge \neg p) \to$ $\mathsf{Z}(p \mathrel{\mathsf{T}} (q \to (p \mathrel{\mathsf{T}} (\neg p)))))$
alt-succession(p,q)	$\mathsf{G}(p \to \mathsf{X}((\neg p) \mathrel{\mathsf{U}} q)) \wedge$ $((\neg q) \mathrel{\mathsf{W}} p) \wedge$ $\mathsf{G}(q \to \widetilde{\mathsf{X}}((\neg q) \mathrel{\mathsf{W}} p))$	$\mathrm{past}(\texttt{alt-response}(p,q)) \wedge$ $\mathrm{past}(\texttt{alt-precedence}(p,q))$
chain-response(p,q)	$\mathsf{G}(p \to \mathsf{X}(q))$	$\neg p \wedge \mathsf{H}(\mathsf{Y}(p) \to q)$
chain-precedence(p,q)	$\mathsf{G}(\mathsf{X}(q) \to p)$	$\mathsf{H}(q \to \mathsf{Z}p)$
chain-succession(p,q)	$\mathsf{G}(p \leftrightarrow \mathsf{X}(q))$	$\mathrm{past}(\texttt{chain-response}(p,q)) \wedge$ $\mathrm{past}(\texttt{chain-precedence}(p,q))$
not-coexistence(p,q)	$\neg(\mathsf{F}(p) \wedge \mathsf{F}(q))$	$\mathsf{H}(\neg p) \vee \mathsf{H}(\neg q)$
neg-succession(p,q)	$\mathsf{G}(p \to \neg\mathsf{F}(q))$	$\mathsf{H}(\neg p) \vee (\neg q) \mathrel{\mathsf{S}} (p \wedge \neg q \wedge \mathsf{ZH}(\neg p))$
neg-chain-succession(p,q)	$\mathsf{G}(p \to \widetilde{\mathsf{X}}(\neg q)) \wedge$ $\mathsf{G}(q \to \widetilde{\mathsf{X}}(\neg p))$	$\mathsf{H}(\mathsf{Y}(p) \to \neg q) \wedge$ $\mathsf{H}(\mathsf{Y}(q) \to \neg p)$

Table 1: DECLARE templates, their LTLf formalization and pastification. Grey cells are a contribution of this paper.

more details on this procedure): if this is the case, then ϕ is realizable and a strategy s can be extracted, otherwise ϕ is unrealizable. The proof concludes by observing that reachability games can be solved in PTIME in the size of the input automaton (De Alfaro, Henzinger, and Kupferman 2007).

We remark that, for a realizable pair of DECLARE specifications, Algorithm 1 synthesizes a strategy for Controller.

At this point, one may wonder whether it could be possible to obtain a lower complexity bound for DECLARE synthesis. However, this task is far from being trivial. In fact, it suffices to look at the DECLARE consistency problem (i.e., checking whether a DECLARE specification is satisfiable), that is subsumed by the DECLARE synthesis problem, studied in (Fionda and Greco 2016; Fionda and Guzzo 2019). These works only demonstrate NP-hardness for DECLARE consistency, but never investigate its completeness. The lack of the completeness result is due to the fact that DECLARE is not closed under Boolean operations (except for the conjunction), which makes it difficult to devise reductions from classical PSPACE-complete or 2EXPTIME-complete prob-

lems, like the tiling problem where disjunctions are key to encode the correct alignment of tiles (van Emde Boas 1996).

Symbolic Reactive Synthesis for DECLARE

We show now a new algorithm for building *linear-size*, *symbolic* DFAs starting from (the pastification of) DECLARE specifications. Such DFAs are then used to obtain a symbolic algorithm for DECLARE realizability. This allows to exploit efficient, performance-tuned tools for solving reachability games that have emerged in the last decade (Jacobs et al. 2017), an essential step towards practical feasibility.

Symbolic DFAs for pure past LTLf formulas

Our approach comprises two steps. Given a DECLARE specification ψ: (i) we obtain a linear-size pastification $\mathrm{past}(\psi)$ as described in **??** ; (ii) we build a linear-size, symbolic DFA $\mathcal{S}$ that recognizes the language of $\mathrm{past}(\psi)$. The second step is carried out through a novel encoding of general $\mathrm{LTLf_P}$ formulas into symbolic DFAs.

Let ϕ be a LTLf$_P$ formula over the alphabet Σ. In the following, we describe how to build a symbolic DFA $\mathcal{S}(\phi) = (X \cup \Sigma, I(X), T(X, \Sigma, X'), F(X, \Sigma))$ s.t. $\mathcal{L}(\mathcal{S}(\phi)) = \mathcal{L}(\phi)$ and $|\mathcal{S}(\phi)|$ is linear in $|\phi|$. We first define the *closure* of ϕ.

Definition 8. *The* closure *of a* LTLf$_P$ *formula ϕ over the alphabet Σ, denoted as $\mathcal{C}(\phi)$, is the smallest set of formulas satisfying the following properties:*
1. *$\phi \in \mathcal{C}(\phi)$, and, for each subformula ϕ' of ϕ, $\phi' \in \mathcal{C}(\phi)$;*
2. *for each $p \in \Sigma$, $p \in \mathcal{C}(\phi)$ if and only if $\neg p \in \mathcal{C}(\phi)$;*
3. *if ϕ_1 S ϕ_2 (resp., ϕ_1 T ϕ_2) is in $\mathcal{C}(\phi)$, then $Y(\phi_1$ S $\phi_2)$ (resp., $Z(\phi_1$ T $\phi_2)$) is in $\mathcal{C}(\phi)$.*

State variables. We denote by $\mathcal{C}_Y(\phi)$ (resp., $\mathcal{C}_Z(\phi)$) the set of formulas of type $Y\phi_1$ (resp., $Z\phi_1$) in $\mathcal{C}(\phi)$. For each formula ψ in $\mathcal{C}_Y(\phi) \cup \mathcal{C}_Z(\phi)$, we introduce a state variable x_ψ that tracks the truth of ψ over any run of the automaton:

$$X := \{x_\psi \mid \psi \in \mathcal{C}_Y(\phi) \cup \mathcal{C}_Z(\phi)\}$$

Formula for the initial states. $I(X)$ describes the initial states as the formula setting all variables x_ψ s.t. ψ is in $\mathcal{C}_Y(\phi)$ (resp., ψ is in $\mathcal{C}_Z(\phi)$) to false (resp., to true):

$$I(X) := \bigwedge_{x_\psi . \psi \in \mathcal{C}_Y(\phi)} \neg x_\psi \wedge \bigwedge_{x_\psi . \psi \in \mathcal{C}_Z(\phi)} x_\psi$$

This forces each formula of type $Y\phi_1$ (resp., of type $Z\phi_1$) to be false (resp., true) at the first state. Note that there is exactly one satisfying assignment to $I(X)$.

Formula for the transition relation. $T(X, \Sigma, X')$ is the conjunction of formulas for the transition relation of each state variable $x \in X$, that we shall define. We first define a normal form for LTLf$_P$ formulas tailored to our needs.

Definition 9. *Let ϕ be a LTLf$_P$ formula over Σ. Its* stepped normal form*, denoted by $\mathrm{snf}(\phi)$, is defined as follows:*

$$
\begin{aligned}
\mathrm{snf}(\ell) &= \ell & \text{where } \ell \in \{p, \neg p\}, \text{ for } p \in \Sigma \\
\mathrm{snf}(\otimes \phi_1) &= \otimes \phi_1 & \text{where } \otimes \in \{Y, Z\} \\
\mathrm{snf}(\phi_1 \otimes \phi_2) &= \mathrm{snf}(\phi_1) \otimes \mathrm{snf}(\phi_2) & \text{where } \otimes \in \{\wedge, \vee\} \\
\mathrm{snf}(\phi_1 \, S \, \phi_2) &= \mathrm{snf}(\phi_2) \vee (\mathrm{snf}(\phi_1) \wedge Y(\phi_1 \, S \, \phi_2)) \\
\mathrm{snf}(\phi_1 \, T \, \phi_2) &= \mathrm{snf}(\phi_2) \wedge (\mathrm{snf}(\phi_1) \vee Z(\phi_1 \, T \, \phi_2))
\end{aligned}
$$

Given an LTLf$_P$ formula ϕ, $\mathrm{ground}(\phi)$ is the formula obtained from ϕ by replacing each formula of type $Y\phi_1$ (resp., $Z\phi_1$) with the state variable $x_{Y\phi_1}$ (resp., $x_{Z\phi_1}$). Let x_ψ be a variable in X, with $\psi := \otimes(\psi_1)$ for some $\otimes \in \{Y, Z\}$ and $\psi_1 \in \mathcal{C}(\phi)$. We use $\mathrm{ground}(\cdot)$ and $\mathrm{snf}(\cdot)$ to define the transition relation of $x_{\otimes(\psi_1)}$ as $x'_{\otimes(\psi_1)} \leftrightarrow \mathrm{ground}(\mathrm{snf}(\psi_1))$. Intuitively, $x'_{\otimes(\psi_1)}$ is true at time point $i+1$ iff $\mathrm{ground}(\mathrm{snf}(\psi_1))$ is true at time point i. For every ψ_1, the transition relations for $x_{Y\psi_1}$ and $x_{Z\psi_1}$ are the same: the value of these state variables differs only in the initial state. In fact, the Y and Z operators have the same non-initial time point semantics.

Formula for final states. The final state formula $F(X, \Sigma)$ captures states in which ϕ holds. Encoded as $F(X, \Sigma) := \mathrm{ground}(\mathrm{snf}(\phi))$, it reflects that every trace reaching, at time point i, a state satisfying $F(X, \Sigma)$, is forced to satisfy ϕ at i.

All in all, $\mathcal{S}(\phi) = (X \cup \Sigma, I(X), T(X, \Sigma, X'), F(X, \Sigma))$ is indeed a symbolic DFA, since: (i) formula $I(X)$ has exactly one satisfying assignment; (ii) formula $T(X, \Sigma, X')$ is of the form $\bigwedge_{x \in X}(x' \leftrightarrow \beta_x(V))$, where $\beta_x(V)$ is a Boolean formula over $V (= X \cup \Sigma)$, for each $x \in X$. The next theorem demonstrates the correctness of our procedure.

Theorem 7. *For any* DECLARE *model ϕ, the automaton $\mathcal{S}(\phi)$ is s.t.: (i) $\mathcal{S}(\phi)$ is symbolic* DFA*; (ii) $\mathcal{L}(\mathcal{S}(\phi)) = \mathcal{L}(\phi)$; (iii) $|\mathcal{S}(\phi)| \in O(|\phi|)$.*

Example 8. *Consider the* precedence(a, b) DECLARE *pattern. Its pastification is $\varphi := H(b \to O(a))$ (cf. Table 1), and the corresponding* DFA *is as follows:*
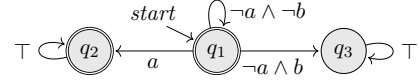
To obtain the symbolic automaton $\mathcal{S}(\varphi)$ for φ, we first compute its stepped normal form: $\mathrm{snf}(\varphi) = (b \to (a \vee YO(a))) \wedge Z(\varphi)$. Since the formula is not expandable anymore, we proceed by defining the set X of state variables as $\{x_{YO(a)}, x_{Z(\varphi)}\}$. The initial and final state formulas of $\mathcal{S}(\phi)$ are $I(X) = \neg x_{YO(a)} \wedge x_{Z(\varphi)}$ and $F(X, \{a, b\}) = \mathrm{snf}(\varphi)$. The transition relation $T(X, \{a, b\}, X')$ consists of the two formulas $x'_{YO(a)} \leftrightarrow \mathrm{snf}(O(a))$ and $x'_{Z(\varphi)} \leftrightarrow \mathrm{snf}(\varphi)$.

Comparison with existing approaches. Our encoding is simpler than the one in (Cimatti et al. 2021; Geatti 2022), since it does not use any counter bit, and largely reduces the number of state variables. For example, in (Cimatti et al. 2021) a state variable is used for each subformula of the initial formula, while here we introduce a state variable only for those of the form $Y\phi$ or $Z\phi$. Moreover, using the notion of *stepped normal form*, the proofs of correctness of our approach are a lot more succinct and easy to understand with respect to those form (Cimatti et al. 2021; Geatti 2022). Clearly, an experimental evaluation needs to be done in order to compare the performance of the two approaches.

Symbolic DECLARE Realizability

We combine DECLARE pastification and the encoding of LTLf$_P$ formulas into linear-size, symbolic DFAs, to ultimately obtain a symbolic version of Algorithm 1 for DECLARE realizability and synthesis. It works as follows. The first step (*i.e.*, pastification) is the same as in Algorithm 1. Once a pure-past formula ϕ is obtained, the algorithm encodes it into a DFA $\mathcal{S}(\phi)$, following the procedure described before. Realizability is then decided by solving a *symbolic reachability game* over $\mathcal{S}(\phi)$, following (De Alfaro, Henzinger, and Kupferman 2007; Jacobs et al. 2017). Differently from classical reachability games, this symbolic version can be solved efficiently through manipulation of the Boolean formulas that define $\mathcal{S}(\phi)$, by computing the *strong preimage* of the transition formula until a fixpoint is reached.

We remark that, since the organization of the Synthesis Competition (Jacobs et al. 2017, 2019) (SYNTCOMP), many optimized tools have been proposed to solve reachability games over symbolic deterministic automata, like, for instance, SafetySynth (Jacobs et al. 2019) and

`Demiurge` (Bloem, Könighofer, and Seidl 2014), that have reached outstanding performance. Using these tools as backends for solving reachability games has thus the potential of making DECLARE reactive synthesis used in practice.

Conclusions

We have introduced realizability and reactive synthesis for DECLARE, a well-established declarative, pattern-based business process modelling language grounded in LTLf. We have shown that DECLARE enjoys the key property of polynomial-size pastification, using it to provide a singly exponential-time algorithm, which contrasts with the 2EXPTIME-completeness of realizability for full LTLf. Notably, by considering the similarity of DECLARE patterns to trajectory constraints in planning (De Giacomo, De Masellis, and Montali 2014), and exploiting their pure-past encoding shown in (De Giacomo, Favorito, and Fuggitti 2022), our approach seamlessly carries over such temporal patterns.

We also provided a fully symbolic version of the previous algorithm, which constitutes a solid basis for the practical applicability of the approach. In fact, since the organization of the SYNTCOMP (Jacobs et al. 2017), many optimized tools have been developed to solve symbolic reachability games, at the core of our symbolic procedure.

We foresee three main lines of future work. First, it is interesting to study algorithmic optimizations (by taking into account, e.g., the structure of DECLARE patterns), ranking of strategies and tighter complexity upper bound of the proposed synthesis procedure. Second, we intend to perform a large-scale experimental evaluation of our symbolic procedure, using different symbolic reachability game solvers and contrasting it with explicit approaches. Third, we intend to study if and how our results carry over different reasoning tasks, such as DECLARE monitoring and automated discovery from execution data, which employ automata-based techniques at their core (Di Ciccio and Montali 2022).

Acknowledgments

References

Artale, A.; Geatti, L.; Gigante, N.; Mazzullo, A.; and Montanari, A. 2023a. A Singly Exponential Transformation of LTL[X, F] into Pure Past LTL. In *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning*, 65–74.

Artale, A.; Geatti, L.; Gigante, N.; Mazzullo, A.; and Montanari, A. 2023b. Complexity of Safety and coSafety Fragments of Linear Temporal Logic. 6236–6244.

Bacchus, F.; and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artif. Intell.*, 116(1-2): 123–191.

Bansal, S.; Li, Y.; Tabajara, L. M.; and Vardi, M. Y. 2020. Hybrid Compositional Reasoning for Reactive Synthesis from Finite-Horizon Specifications. In *Proc. of AAAI 2020*, 9766–9774. AAAI Press.

Bloem, R.; Könighofer, R.; and Seidl, M. 2014. SAT-based synthesis methods for safety specs. In *Proc. of VMCAI*, 1–20. Springer.

Buchi, J. R.; and Landweber, L. H. 1990. Solving sequential conditions by finite-state strategies. In *The Collected Works of J. Richard Büchi*, 525–541. Springer.

Burch, J. R.; Clarke, E. M.; McMillan, K. L.; Dill, D. L.; and Hwang, L.-J. 1992. Symbolic model checking: 1020 states and beyond. *Information and computation*, 98(2): 142–170.

Camacho, A.; Baier, J. A.; Muise, C. J.; and McIlraith, S. A. 2018. Finite LTL Synthesis as Planning. In *Proc. of ICAPS 2018*, 29–38. AAAI Press.

Chandra, A. K.; Kozen, D.; and Stockmeyer, L. J. 1981. Alternation. *J. ACM*, (1): 114–133.

Church, A. 1962. Logic, arithmetic, and automata. In *Proc. of ICM*, volume 1962, 23–35.

Cimatti, A.; Geatti, L.; Gigante, N.; Montanari, A.; and Tonetta, S. 2021. Extended bounded response LTL: a new safety fragment for efficient reactive synthesis. *Formal Methods in System Design*, 1–49.

De Alfaro, L.; Henzinger, T. A.; and Kupferman, O. 2007. Concurrent reachability games. *Theoretical Computer Science*, 386(3): 188–217.

De Giacomo, G.; De Masellis, R.; Maggi, F. M.; and Montali, M. 2022. Monitoring Constraints and Metaconstraints with Temporal Logics on Finite Traces. *ACM Trans. Softw. Eng. Methodol.*, 31(4): 68:1–68:44.

De Giacomo, G.; De Masellis, R.; and Montali, M. 2014. Reasoning on LTL on Finite Traces: Insensitivity to Infiniteness. In *Proc. of AAAI 2014*, 1027–1033. AAAI Press.

De Giacomo, G.; De Masellis, R.; and Montali, M. 2014. Reasoning on LTL on Finite Traces: Insensitivity to Infiniteness. In *Proc. of (AAAI-14)*, 1027–1033. AAAI Press.

De Giacomo, G.; Di Stasio, A.; Fuggitti, F.; and Rubin, S. 2021. Pure-past linear temporal and dynamic logic on finite traces. In *Proc. of IJCAI*, 4959–4965.

De Giacomo, G.; Favorito, M.; and Fuggitti, F. 2022. Planning for Temporally Extended Goals in Pure-Past Linear Temporal Logic: A Polynomial Reduction to Standard Planning. *CoRR*, abs/2204.09960.

De Giacomo, G.; and Vardi, M. Y. 2013. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In Rossi, F., ed., *Proc. of IJCAI 2013*, 854–860. IJCAI/AAAI.

De Giacomo, G.; and Vardi, M. Y. 2015. Synthesis for LTL and LDL on Finite Traces. In *Proc. of AAAI 2015*, 1558–1564. AAAI Press.

Di Ciccio, C.; and Montali, M. 2022. Declarative Process Specifications: Reasoning, Discovery, Monitoring. In van der Aalst, W. M. P.; and Carmona, J., eds., *Process Mining Handbook*, LNBIP, 108–152. Springer.

Dumas, M.; Rosa, M. L.; Mendling, J.; and Reijers, H. A. 2018. *Fundamentals of Business Process Management, Second Edition*. Springer.

Dwyer, M. B.; Avrunin, G. S.; and Corbett, J. C. 1999. Patterns in Property Specifications for Finite-State Verification. In *Proc. of ICSE 1999*, 411–420. ACM.

Fionda, V.; and Greco, G. 2016. The complexity of LTL on finite traces: Hard and easy fragments. In *Proc. of AAAI*, volume 30.

Fionda, V.; and Greco, G. 2018. LTL on Finite and Process Traces: Complexity Results and a Practical Reasoner. *J. Artif. Intell. Res.*, 63: 557–623.

Fionda, V.; and Guzzo, A. 2019. Control-Flow Modeling with Declare: Behavioral Properties, Computational Complexity, and Tools. *IEEE Transactions on Knowledge and Data Engineering*, 32(5): 898–911.

Geatti, L. 2022. *Temporal Logic Specifications: Expressiveness, Satisfiability And Realizability*. Phd thesis, University of Udine.

Geatti, L.; Gigante, N.; and Montanari, A. 2021. BLACK: A Fast, Flexible and Reliable LTL Satisfiability Checker. In *Proc. of OVERLAY*, volume 2987 of *CEUR Workshop Proceedings*, 7–12. CEUR-WS.org.

Geatti, L.; Gigante, N.; Montanari, A.; and Venturato, G. 2021. Past Matters: Supporting LTL+Past in the BLACK Satisfiability Checker. In *Proceedings of the 28th International Symposium on Temporal Representation and Reasoning*.

Geatti, L.; Montali, M.; and Rivkin, A. 2022. Reactive Synthesis for DECLARE via symbolic automata.

Geatti, L.; Montali, M.; and Rivkin, A. 2023. Foundations of Collaborative DECLARE. In Francescomarino, C. D.; Burattin, A.; Janiesch, C.; and Sadiq, S. W., eds., *Proc. of BPM*, volume 490 of *Lecture Notes in Business Information Processing*, 55–72. Springer.

Gerevini, A.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artif. Intell.*, 173(5-6): 619–668.

Giacomo, G. D.; Favorito, M.; Li, J.; Vardi, M. Y.; Xiao, S.; and Zhu, S. 2022. LTLf Synthesis as AND-OR Graph Search: Knowledge Compilation at Work. In Raedt, L. D., ed., *Proc. of IJCAI 2022*, 2591–2598. ijcai.org.

Harel, D.; and Pnueli, A. 1984. On the Development of Reactive Systems. In Apt, K. R., ed., *Proc. of LMCS 1984*, NATO ASI Series, 477–498. Springer.

Hopcroft, J. E.; Motwani, R.; and Ullman, J. D. 2001. Introduction to automata theory, languages, and computation. *Acm Sigact News*, 32(1): 60–65.

Jacobs, S.; Bloem, R.; Brenguier, R.; Ehlers, R.; Hell, T.; Könighofer, R.; Pérez, G. A.; Raskin, J.-F.; Ryzhyk, L.; Sankur, O.; et al. 2017. The first reactive synthesis competition (SYNTCOMP 2014). *International journal on software tools for technology transfer*, 19(3): 367–390.

Jacobs, S.; Bloem, R.; Colange, M.; Faymonville, P.; Finkbeiner, B.; Khalimov, A.; Klein, F.; Luttenberger, M.;

Meyer, P. J.; Michaud, T.; Sakr, M.; Sickert, S.; Tentrup, L.; and Walker, A. 2019. The 5th Reactive Synthesis Competition (SYNTCOMP 2018): Benchmarks, Participants & Results. *CoRR*, abs/1904.07736.

Lichtenstein, O.; Pnueli, A.; and Zuck, L. 1985. The glory of the past. In *Proc. of LoP*, 196–218. Springer.

Maggi, F. M.; Montali, M.; and Peñaloza, R. 2020. Temporal Logics Over Finite Traces with Uncertainty. In *Proc. of AAAI 2020*, 10218–10225. AAAI Press.

Maler, O.; Nickovic, D.; and Pnueli, A. 2005. Real time temporal logic: Past, present, future. In *Proc. of FORMATS 2005*, 2–16. Springer.

McMillan, K. L. 1993. Symbolic model checking. In *Symbolic Model Checking*, 25–60. Springer.

Montali, M. 2010. *Specification and verification of declarative open interaction models: a logic-based approach*. Springer Science & Business Media.

Montali, M.; Pesic, M.; van der Aalst, W. M. P.; Chesani, F.; Mello, P.; and Storari, S. 2010. Declarative specification and verification of service choreographiess. *ACM Trans. Web*, 4(1): 3:1–3:62.

Pesic, M.; Schonenberg, H.; and van der Aalst, W. M. P. 2007. DECLARE: Full Support for Loosely-Structured Processes. In *Proc. of EDOC 2007*, 287–300. IEEE Computer Society.

Pesic, M.; and van der Aalst, W. M. P. 2006. A Declarative Approach for Flexible Business Processes Management. In *Proc. of BPM*, LNCS, 169–180. Springer.

Pnueli, A. 1977. The temporal logic of programs. In *Proc. of SFSC*, 46–57. IEEE.

Pnueli, A.; and Rosner, R. 1989a. On the Synthesis of a Reactive Module. In *Proceedings of POPL'89*, 179–190. ACM Press.

Pnueli, A.; and Rosner, R. 1989b. On the synthesis of an asynchronous reactive module. In *Proc. of ICALP*, 652–671. Springer.

Reichert, M.; and Weber, B. 2012. *Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies*. Springer.

Rosner, R. 1992. *Modular synthesis of reactive systems*. Ph.D. thesis, PhD thesis, Weizmann Institute of Science.

van Emde Boas, P. 1996. The Convenience of Tilings. 1–33.

Vardi, M. Y. 1995. An Automata-Theoretic Approach to Fair Realizability and Synthesis. In *Proc. of CAV*, LNCS, 267–278. Springer.

Zhu, S.; and De Giacomo, G. 2022. Act for Your Duties but Maintain Your Rights. In Kern-Isberner, G.; Lakemeyer, G.; and Meyer, T., eds., *Proc. of KR 2022*.

Zhu, S.; Giacomo, G. D.; Pu, G.; and Vardi, M. Y. 2020. LTLf Synthesis with Fairness and Stability Assumptions. In *Proc. of AAAI 2020*, 3088–3095. AAAI Press.

Zhu, S.; Pu, G.; and Vardi, M. Y. 2019. First-Order vs. Second-Order Encodings for LTL$_f$-to-Automata Translation. In *Proc. of TAMC 2019*, LNCS, 684–705. Springer.