Probabilistic Temporal Reasoning Using Superposition Semantics

FABRIZIO M. MAGGI and MARCO MONTALI, Free University of Bozen-Bolzano, Bolzano, Italy
RAFAEL PEÑALOZA, University of Milano-Bicocca, Milano, Italy

Temporal logics over finite traces have recently seen wide application in a number of areas, from business process modelling, monitoring and mining to planning and decision-making. However, real-life dynamic systems contain a degree of uncertainty which cannot be handled with classical logics. We thus propose a new probabilistic temporal logic over finite traces using superposition semantics, where all possible evolutions are possible, until observed. We study the properties of the logic and provide automata-based mechanisms for deriving probabilistic inferences from its formulas. We then study a fragment of the logic with better computational properties. Notably, formulas in this fragment can be discovered from event log data using off-the-shelf existing declarative process discovery techniques.

CCS Concepts: • **Theory of computation** → **Modal and temporal logics**; **Automated reasoning**; *Tree languages*;

Additional Key Words and Phrases: temporal logic, probabilistic reasoning, process modelling

ACM Reference format:
Fabrizio M. Maggi, Marco Montali, and Rafael Peñaloza. 2025. Probabilistic Temporal Reasoning Using Superposition Semantics. *ACM Trans. Comput. Logic* 26, 2, Article 9 (March 2025), 26 pages.
https://doi.org/10.1145/3714427

1 Introduction

Linear temporal logic (LTL) is one of the most important formalisms to declaratively specify and reason about the evolution of systems and processes [4]. Traditionally, LTL adopts a linear, infinite model of time where formulas are interpreted over infinite traces; i.e., sequences of timepoints. In recent years, increasing attention has been given to a different version of the logic, called *LTL over finite traces* or LTL_f [13], which adopts instead a finite-trace semantics. From the modelling point of view, LTL_f matches settings where each execution of the system is eventually expected to end (even though there is no explicit bound on the number of steps required to reach the termination point). From the reasoning point of view, the automata-theoretic characterisation of LTL_f relies on classical finite-state automata [10, 13], which are easier to manipulate than the automata on infinite words that underlie the reasoning techniques in LTL, and pave the way for the development of robust

This work was partially supported by the MUR for the Department of Excellence DISCo at the University of Milano-Bicocca (RegAInS) and under the PRIN project PINPOINT Prot. 2020FNEB27; by the UNIBZ project ADAPTERS; and by the NextGenerationEU FAIR PE0000013 projects MAIPM (CUP C63C22000770006) and AMAR (CUP D53C22002380006).
Authors' Contact Information: Fabrizio M. Maggi, Free University of Bozen-Bolzano, Bolzano, Italy; e-mail: maggi@inf.unibz.it; Marco Montali, Free University of Bozen-Bolzano, Bolzano, Italy; e-mail: montali@inf.unibz.it; Rafael Peñaloza (corresponding author), University of Milano-Bicocca, Millano, Italy; e-mail: rafael.penaloza@unimib.it.

ACM 1557-945X/2025/3-ART9
https://doi.org/10.1145/3714427

and efficient reasoning techniques. In fact, LTL_f and its extensions have been widely employed in a number of application domains relevant for AI: from declarative business process modelling [35, 39], monitoring [9, 33] and mining [11, 30], to planning [12, 19] and decision-making [5], among many others.

When considering real-world processes, uncertainty (which is inexpressible in classical logics) is unavoidable, mainly due to lack of knowledge—epistemic uncertainty—or to noise from the environment—stochastic uncertainty [42]. For example, some pieces needed in a step may be defective, external events may delay a service and the outcome of an action may depend on various implicit factors, which may not be observable by the process. Handling these scenarios requires a logical formalism capable of expressing uncertainty. Surprisingly, to the best of our knowledge no probabilistic extension of LTL_f has been considered so far.

Several probabilistic extensions of infinite-time temporal logics have been proposed over the years. To cover the wide spectrum of approaches, we mention some notable examples, while referring the interested reader to two detailed overviews [15, 27]. A general, modal-based **probabilistic temporal logic (PLTL)** for reasoning about actions and uncertainty was proposed in [48]. In that work, uncertainty is expressed through labelled modalities, which are then combined to produce adequate probability measures. Closer to our view on LTL are [28, 36, 38]. The approach in [28] introduces a probability distribution over the timepoints where an event may be observed as time evolves. Similarly, [36] introduces a distributed temporal logic which directly extends LTL. Perhaps the closest to our work is [38] which uses finitely-additive probabilities, while considering also first-order expressions. One common theme among these works is that the complex interactions between probabilities and time calls for special syntactic or semantic restrictions in the logic, which do not directly carry over to the finite-trace setting. For instance, the probability distribution in [28] would need to be bounded in order to be finite, which is a much stronger constraint than what LTL_f requires. Similarly, the semantics from [38] are only well-defined over infinite time structures.

A different strategy to deal with uncertainty evolution over time is to use Markov chains [34]. In particular, PCTL [21] associates to each state of a Markov chain a propositional valuation, which refers to the observable behaviour of the system at that state. From a given state, the Markov chain defines a probability distribution over the possible successive states. In particular, at every step the system makes a 'choice' of the following state, and cannot go back, thus affecting the full future probability computation, also of the unobservable elements. To the best of our knowledge, finite-time variants of PCTL have not been studied, although recent work has shown that the problem of finding out whether a PCTL formula has a finite model is actually undecidable [7].

We are interested in a probabilistic extension of LTL_f (over finite traces) which does not impose strong syntactic limitations, and where each of the possible executions remains so as long as no (observable) evidence to the contrary is available. We thus propose a new probabilistic extension of LTL_f, called PLTL_f, that essentially predicates over the possible evolutions of a trace. The main novelty of PLTL_f lies in its *superposition* semantics, where every evolution is possible (with different probabilities) until it is observed. This semantics accommodates a seamless interaction of probabilities and time that was not possible in previous formalisms, and elegantly fits over finite traces. PLTL_f is a direct generalisation of LTL_f: PLTL_f formulas without probabilistic constructors are in fact LTL_f formulas. PLTL_f adequately describes probabilistic temporal or dynamic properties of process executions; e.g., it can express that a shipped package will eventually reach its destination with probability 0.95, or that a machine will fail in the next 100 timepoints with probability below 0.001, among other probabilistic temporal statements. It differs from PCTL in that there is no specific state choice at every timepoint, but rather all choices that do not contradict the observations so far remain possible (with different probabilities). This difference becomes more explicit when we compute trace probabilities in Section 4.2.

While PLTL_f is of interest in any situation that requires to handle finite linear-time executions with uncertainty, our main motivation come from the field of declarative process modelling. In this setting, one can use PLTL_f formulas to descrive the knowledge of how the process should work (restricting the applicable traces), while having the flexibility of dealing with uncertainty. This is a natural choice, as LTL_f has been successfully used to describe deterministic process models.

Our second main contribution is an investigation of the logical and computational properties of PLTL_f, introducing automata-based algorithms for deciding satisfiability of PLTL_f formulas and for computing the most likely executions of a system described in this logic. These core reasoning services provide the basis for sophisticated, domain-specific tasks such as a probabilistic version of conformance checking [6] and (prefix) monitoring [33]. Unsurprisingly, due to the intertwined connection of temporal and probabilistic constructors, handling PLTL_f formulas becomes ExpTime-hard. This leads us to our third contribution: a study of a fragment of PLTL_f, called PLTL_f^0, where the complexity of reasoning falls to PSpace in the length of the formula, matching the classical LTL_f case. Notably, formulas in this fragment can be discovered from event log data using off-the-shelf existing declarative process discovery techniques [30]. Although we motivate PLTL_f^0 through a probabilistic extension of the Declare process modelling language [39], this does not mean that full PLTL_f is not relevant for process modelling as well. Indeed, as we argue at the end of Section 5, there are some properties which require the full power of PLTL_f to be correctly expressed.

This manuscript extends the published work [31] by including all the proofs missing in the conference version in full detail, and extending the explanations and intuitions of the constructions through a detailed example. In addition, it provides a deeper analysis on the applications and uses of the formalism.

2 Preliminaries

We first briefly introduce the fundamental notions from tree automata and weighted automata, assuming a basic knowledge of formal languages. For more details, we point the interested readers to [8, 17].

2.1 Tree Automata

We start by introducing the basic tree-related notation that we use in this article. A *tree* is a set of words of natural numbers $T \subseteq \mathbb{N}^*$, which is closed under prefixes and preceding siblings; that is, if $wi \in T$ for some $i \in \mathbb{N}$, then $w \in T$, and $wj \in T$ for all $1 \leq j \leq i$. A tree is *finite* if its cardinality is finite. Each finite tree T has a maximum number $k \in \mathbb{N}$ (called the *width*) such that $wk \in T$ for some $w \in \mathbb{N}^*$. The empty word ε is the *root*, and a *leaf* is a node $w \in T$ such that $w1 \notin T$. A *labelling* of the tree T on a set Σ is a mapping $T \to \Sigma$. A tree which has been associated to a labelling is called a *labelled tree*. A *branch* of the tree T is a sequence $w_1, \ldots, w_n$ of nodes of T such that $w_1 = \varepsilon$, w_n is a leaf node, and for every $i, 1 \leq i < n$, $w_{i+1} = w_i m$ for some $m \in \mathbb{N}$. With a slight abuse of terminology, whenever a tree T is labelled, we call *branch* also the sequence of labels of a branch.

Definition 2.1 (Tree Automata). A k-ary *tree automaton* is a tuple $\mathcal{A} = (Q, \Delta, I, F)$ where Q is a finite set of *states*, $I, F \subseteq Q$ are sets of states whose elements are called *initial* and *final* states, respectively, and $\Delta \subseteq Q \times \bigcup_{i \leq k} Q^i$ is the *transition relation*. A *run* of $\mathcal{A}$ on the tree T of width k is a labelling $\rho : T \to Q$ such that $\rho(\varepsilon) \in I$ and for every $w \in T$ and $n \in \mathbb{N}$, if $wn \in T$ but $w(n{+}1) \notin T$, then $(\rho(w), \rho(w1), \ldots, \rho(wn)) \in \Delta$. Such a run is called *successful* if for every leaf node $w \in T$, it holds that $\rho(w) \in F$; in that case, we say that $\mathcal{A}$ *accepts* the tree T. The *language accepted* by $\mathcal{A}$ is the set $\mathcal{L}(\mathcal{A})$ of finite trees for which there is a successful run of $\mathcal{A}$. The *emptiness problem* asks whether $\mathcal{L}(\mathcal{A}) = \emptyset$.

Importantly, this definition introduces what is often called a *looping* automaton, where the transitions are not labelled. In practice, the scope of looping automata is only to analyse the structure of the trees it receives as input [2, 22, 23].

The emptiness problem of k-ary tree automata can be decided in time $O(|Q|^{k+2})$ [47] through a computation of so-called *good states*. In a nutshell, a good state is one from which it is possible to construct a labelled subtree that satisfies the transition relation Δ and all its leafs are labelled with a final state. The importance of good states relies on the fact that if a run ρ labels all its leafs as good states, then ρ can be extended to a successful run. In particular, if there is an initial good state, then exists a successful run, and hence $\mathcal{L}(\mathcal{A}) \neq \emptyset$. The converse is also true: if $\mathcal{A}$ accepts at least one tree, then there exists an initial state which is good.

The class $\mathsf{good}(\mathcal{A})$ of good states of the automaton $\mathcal{A} = (Q, \Delta, I, F)$ is formally defined in a recursive manner. First, $F \subseteq \mathsf{good}(\mathcal{A})$; that is, every final state is good. Then, a state $q \notin F$ is good iff there exist $q_1, \ldots, q_k \in \mathsf{good}(\mathcal{A})$ such that $(q, q_1, \ldots, q_k) \in \Delta$. This definition suggests an obvious iterative procedure for computing $\mathsf{good}(\mathcal{A})$: starting from the class of final states, add any state that has at least one transition leading to *only* good states. This iteration reaches a fixpoint after checking the transitions of each state at most $|Q|$ times. As there are at most $|Q|^{k+1}$ transitions, the set of good states is computable in time $O(|Q|^{k+2})$. The following proposition follows directly from the construction of the set $\mathsf{good}(\mathcal{A})$.

Proposition 2.2. *The tree automaton $\mathcal{A}$ is not empty iff there is an initial state that belongs to* $\mathsf{good}(\mathcal{A})$.

The states that are *not* good (from now on called *bad* states) are those from which it is impossible to complete a successful run. For instance, a non-final state which has no outgoing transitions is necessarily bad: if it appears in a run ρ, it must be as the label of a leaf node, but as it is not final, ρ cannot be successful. Once that we have identified these bad states, we can safely remove them from the automaton without affecting its behaviour.

Given an automaton $\mathcal{A}$, its *reduced automaton* $\ddot{\mathcal{A}}$ is obtained by removing all bad states from $\mathcal{A}$. More formally, $\ddot{\mathcal{A}} := (\mathsf{good}(\mathcal{A}), \ddot{\Delta}, \ddot{I}, F)$, where

- $\ddot{\Delta} := \Delta \cap \mathsf{good}(\mathcal{A}) \times \bigcup_{i \leq k} \mathsf{good}(\mathcal{A})^i$ and
- $\ddot{I} := I \cap \mathsf{good}(\mathcal{A})$.

Importantly, the set of final states does not change, because $F \subseteq \mathsf{good}(\mathcal{A})$. It follows directly that $\mathcal{A}$ and $\ddot{\mathcal{A}}$ accept the same language, and hence $\mathcal{L}(\ddot{\mathcal{A}}) \neq \emptyset$ iff $\ddot{I} \neq \emptyset$.

An alternative emptiness test constructs a successful run top-down. It guesses an initial state to label the root node, and iteratively guesses transitions for every node not labelled with a final state. Through a depth-first construction, the algorithm preserves in memory only one branch at a time, together with the information of the chosen transitions. Since every branch can be restricted to depth $|Q|$ (an indirect consequence of Proposition 2.2), the space requirement for this process is in $O(|Q| \cdot k)$ [2].

These properties will come on handy when we analyse the complexity of reasoning in PLTL$_f$ at the end of Section 3 and in Section 4. We note that a tighter upper bound for deciding automata emptiness—specially if the transition relation is sparse—is given by $O(|Q| \cdot |\Delta|)$. The reason we do not choose this expression is because we are more interested in computing the behaviour of a weighted automaton (see the next subsection), where all transitions are assumed to exist. At that point, both measures become equivalent.

2.2 Weighted Automata

The definition of weighted automata [17] requires an algebraic structure known as a *semiring* defined through two binary operators satisfying some basic properties. For the scope of this article,

we use only one specific semiring known as the *probabilistic semiring*, and thus consider only the instance of weighted automata over this semiring. In this case, the automaton will read strings (words) rather than trees.

The *probabilistic semiring* is the algebraic structure $\mathbb{P} = ([0,1], \max, \times, 0, 1)$ where $[0,1]$ is the unit interval of real numbers and max and $\times$ refer to the standard maximum and product operators over real numbers. Note that 0 and 1 are the identities for max and $\times$, respectively, and that $\times$ distributes over max. Weighted automata generalise regular automata by "charging" a cost to each possible transition. Rather than accepting or rejecting a word ω, the automaton assigns to it a final weight, that depends on the costs of all possible runs over ω. As with tree automata before, we consider only the simpler case of *looping* automata, which does not assign labels to the transitions.

Definition 2.3 (Weighted Automata). A *weighted automaton* is a tuple $\mathcal{A} = (Q, \mathsf{in}, \mathsf{wt}, F)$ where Q is a finite set of *states*, $F \subseteq Q$ is the set of *final states*, $\mathsf{in} : Q \to [0,1]$ is the *initialization function* and $\mathsf{wt} : Q \times Q \to [0,1]$ is the *weight function*. A *run* of $\mathcal{A}$ is a finite sequence $\rho = q_0, q_1, \ldots, q_n$ with $q_n \in F$; $\mathsf{run}(\mathcal{A})$ denotes the set of all runs of $\mathcal{A}$. The weight of $\rho = q_0 \ldots, q_n \in \mathsf{run}(\mathcal{A})$ is $\mathsf{wt}(\rho) := \prod_{i=0}^{n-1} \mathsf{wt}(q_i, q_{i+1})$. The *behaviour* of $\mathcal{A}$ is $\|\mathcal{A}\| := \max_{\rho \in \mathsf{run}(\mathcal{A})} \mathsf{in}(q_0) \cdot \mathsf{wt}(\rho)$.[1]

To compute the behaviour of the weighted automaton $\mathcal{A}$, we adapt the emptiness test for unweighted automata[2] to consider the weights of the transitions computing a function $\mathsf{w} : Q \to [0,1]$, where $\mathsf{w}(q)$ is the maximum weight of all runs starting in q. Initialize $\mathsf{w}_0(q) = 1$ if $q \in F$ and $\mathsf{w}_0(q) = 0$ if $q \notin F$. For each $i \in \mathbb{N}$ iteratively compute $\mathsf{w}_{i+1}(q) := \max\{\mathsf{w}_i(q), \max_{q' \in Q} \mathsf{wt}(q, q')\mathsf{w}_i(q')\}$. Note that the weight associated to each state is monotonically increasing with each iteration. After linearly many steps on the number of states of $\mathcal{A}$, we are guaranteed to reach a fixpoint where $\mathsf{w}_i \equiv \mathsf{w}_{i+1}$. If $\mathsf{w} := \mathsf{w}_i$, then $\|\mathcal{A}\| = \max_{q \in Q} \mathsf{in}(q)\mathsf{w}(q)$.

In fact, it is easy to see that $w_i(q)$ is the maximum weight for all runs of $\mathcal{A}$ that have length at most i. In addition, any run of length greater than $n := |Q|$ must necessarily traverse a state at least twice and can thus be simplified into a shorter run that does not repeat nodes and has a higher weight, because the product over $[0,1]$ is monotonically decreasing. In particular, $\mathsf{w}_{n+1} \equiv \mathsf{w}_{n+2}$. For full details see e.g., [3].

After these preliminaries, we now introduce a PLTL over finite traces, which we call PLTL_f.

3 The Probabilistic Temporal Logic PLTL_f

PLTL_f extends the LTL on finite traces LTL_f [13], with a probabilistic constructor expressing uncertainty about the evolution of traces. The only syntactic difference between LTL_f and PLTL_f is this new constructor, which bounds the probability of observing a given behaviour in the future. Formally, PLTL_f formulas are built by the following syntactic rule where a is a propositional variable, $p \in [0,1]$, and $\bowtie \in \{\leq, \geq, <, >\}$:

$$\varphi ::= a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \varphi \,\mathcal{U}\, \varphi \mid \circledcirc_{\bowtie p}\varphi.$$

Intuitively, $\circledcirc_{\bowtie p}\varphi$ means that, at the next point in time, φ holds with probability $\bowtie p$. Importantly, such a formula also provides information about the probability of $\neg\varphi$. For instance, if the probability of a formula φ (at the next timepoint) is greater or equal to 0.7, then it follows that the probability of $\neg\varphi$ will be at most $1 - 0.7 = 0.3$. To formalise this intuition, we use tree-shaped interpretations providing a class of alternatives branching into the future; in the previous example, one branch considers φ and another considers $\neg\varphi$. As an analogy for uncertainty in physical systems, we call this approach the *superposition semantics*.

[1] As standard, $\prod_{x \in \emptyset} x = 1$ and $\max_{x \in \emptyset} x = 0$.

[2] A regular string automaton is a special case of a tree automaton, where all transitions have arity 1. The emptiness test that checks for good states can be directly applied to this setting.

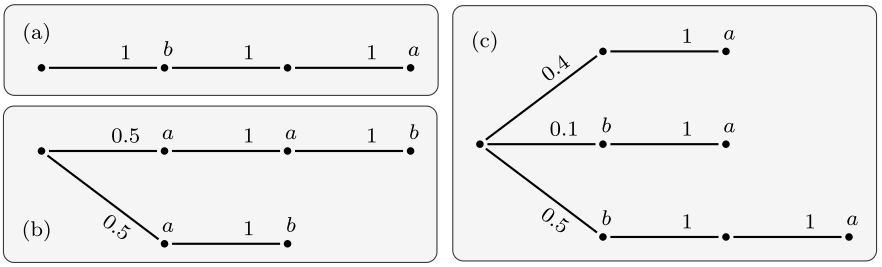

Fig. 1. Three models of the formula $\phi_0 := \circledcirc_{\leq 0.5} \bigcirc a \wedge \circledcirc_{\geq 0.6}(a \,\mathcal{U}\, b)$ from Example 3.3.

Definition 3.1 (Superposition Semantics). A *PLTL$_f$ interpretation* is a triple $I = (T, \cdot^I, P)$, where T is a finite tree, $\cdot^I$ is a labelling of T on the set of propositional valuations,[3] and $P : T \setminus \{\varepsilon\} \to [0, 1]$ is such that for each non-leaf node $w \in T$, it holds that $\sum_{wi \in T} P(wi) = 1$. Satisfiability of a formula in a tree node is defined inductively, extending the LTL$_f$ semantics. For an interpretation $I = (T, \cdot^I, P)$ and $w \in T$:

— $I, w \models a$ iff $a \in w^I$;
— $I, w \models \neg\varphi$ iff $I, w \not\models \varphi$;
— $I, w \models \varphi \wedge \psi$ iff $I, w \models \varphi$ and $I, w \models \psi$;
— $I, w \models \bigcirc\varphi$ iff w is not a leaf node and for all $i \in \mathbb{N}$, if $wi \in T$ then $I, wi \models \varphi$;
— $I, w \models \varphi \,\mathcal{U}\, \psi$ iff either (i) $I, w \models \psi$ or (ii) $I, w \models \varphi$ and $I, w \models \bigcirc(\varphi \,\mathcal{U}\, \psi)$ and
— $I, w \models \circledcirc_{\bowtie p}\varphi$ iff $\sum_{wi \in T; I, wi \models \varphi} P(wi) \bowtie p$.[4]

I is a *model* of ϕ if $I, \varepsilon \models \phi$. ϕ is *satisfiable* if it has a model.

Note in particular that, similarly to LTL$_f$, since trees are finite, the formula $\varphi \,\mathcal{U}\, \psi$ requires that ψ is eventually satisfied in all branches evolving from the current point in time. Another important element of our semantics is that the uncertainty refers always to the future of the trace, never to the current point in time; we try to make this apparent through the choice of the notation of this operator $\circledcirc$. In addition, every leaf node w of a PLTL$_f$ interpretation satisfies any formula of the form $\circledcirc_{\leq p}\psi$ or $\circledcirc_{<q}\psi$ with $p \in [0, 1]$ and $q \in (0, 1]$: the probability of observing ψ in the next timepoint is 0, due to the lack of successors of leaf nodes. We use all the standard abbreviations from LTL$_f$; in particular, $\Diamond\varphi \equiv \top U \varphi$, where $\top$ stands for any tautology, and $\Box\varphi \equiv \neg\Diamond\neg\varphi$, which express that φ will *eventually* hold and that φ holds *always*, respectively. The following example highlights the main features of the superposition semantics.

Example 3.2. Consider the PLTL$_f$ formula $\phi_0 := \circledcirc_{\leq 0.5} \bigcirc a \wedge \circledcirc_{\geq 0.6}(a \,\mathcal{U}\, b)$. Figure 1 depicts three different models of this formula. As a convention, we number siblings in these trees from top to bottom.

Figure 1(a) corresponds to the interpretation $I = (T, \cdot^I, P)$ where:

— $T = \{\varepsilon, 1, 11, 111\}$;
— $P(w) = 1$ for each $w \neq \varepsilon$ and
— $\epsilon^I = 11^I = \emptyset$, $1^I = \{b\}$ and $111^I = \{a\}$.

Following the semantics, we can see that $I, 1 \models b$ (and hence $I, 1 \models a \,\mathcal{U}\, b$) and that $I, 2 \not\models a$, which implies $I, 1 \not\models \bigcirc a$. Overall, this means that from the timepoint ε, the probabilities of observing

[3] As usual, we describe a propositional valuation by the set of variables it makes true.
[4] If w has no successor node that satisfies φ—in particular if w is a leaf node—then $\sum_{wi \in T; I, wi \models \varphi} P(wi) = 0$ as standard in recursive operations.

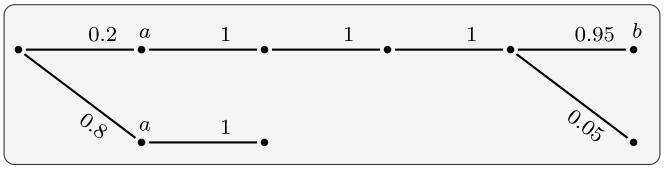

Fig. 2. A model of the formula $\phi_2 := \circledcirc_{\leq 0.2}(a \wedge \circledcirc_{\geq 0.9} \Diamond b)$ from Example 3.3.

$a \,\mathcal{U}\, b$ and $\bigcirc a$ at the next timepoint are, respectively, 1 and 0, which means that I is indeed a model of ϕ_0.

Similarly, one can see that in the interpretation depicted in Figure 1(b), $I, 1 \models a \,\mathcal{U}\, b$ and $I, 2 \models a \,\mathcal{U}\, b$, which implies that $I, \varepsilon \models \circledcirc_{\geq 0.6} a \,\mathcal{U}\, b$. On the other hand, although $I, 1$ satisfies the formula $\bigcirc a$, $I, 2$ does not, and thus $I, \varepsilon \models \circledcirc_{\leq 0.5} \bigcirc a$. We leave as an exercise to the interested reader to verify that the interpretation in Figure 1(c) is indeed a model of ϕ_0 but not of the formula $\circledcirc_{\leq 0.4} \bigcirc a$. Clearly, many other models can be constructed for the same formula.

In general, the probabilistic constructor $\circledcirc$ allows additional flexibility for constructing interpretations satisfying a given PLTL$_f$ formula. And yet not all PLTL$_f$ formulas have a model. As a simple example consider the formula $\phi_1 := \circledcirc_{\geq 0.5} a \wedge \circledcirc_{\geq 0.6} \neg a$. No interpretation allows a and $\neg a$ to hold with probability 0.5 and 0.6, respectively at any given timepoint. Hence, ϕ_1 is unsatisfiable.

Example 3.3. Consider now the formula $\phi_2 := \circledcirc_{\leq 0.2}(a \wedge \circledcirc_{\geq 0.9} \Diamond b)$. This forms a pattern which could be encountered in a business process modelling scenario: if we consider the variable b to stand for return of an order, the subformula $\circledcirc_{\geq 0.9} \Diamond b$ states that it is very likely (with probability at least 0.9) that an item is eventually returned; i.e., it refers to items with a high risk of being returned. If a stands for purchase, then ϕ_2 expresses that it is very unlikely that items with a high-return risk are purchased. A possible model of this formula is depicted in Figure 2. Note from this example that it is possible to purchase the item without any issues (the lower branch at the beginning of the model) or to purchase it and after some time either return it (upper branch) or keep it (lower branch at the end). In particular, the probability of *not* returning the item within this model is 0.81.

As usual, the main reasoning task in PLTL$_f$ is to check satisfiability of its formulas. In comparison to LTL$_f$, this reasoning problem becomes harder in PLTL$_f$ due to the need to verify the compatibility of the potential future steps w.r.t. their probabilities, and in particular analyse the branching allowed by the superposition semantics. This should not be surprising since, as argued in the introduction, the combination of uncertainty and time is known to require careful reasoning analysis. Based on the automata-based approach originally developed for LTL [46] (but adapted to the finite case, requiring no Büchi condition), we construct a tree automaton accepting a class of models for a given formula ϕ. For satisfiability, we can ignore the specific probabilities used, as long as they are *compatible* in a sense that will be formalised below, but which allows us to consider unweighted tree automata only. Before presenting the formal construction of such an automaton, some definitions are necessary.

Definition 3.4 (Atom). Let ϕ be a PLTL$_f$ formula. The *closure of subformulas* of ϕ is the smallest set $\mathsf{csub}(\phi)$ of PLTL$_f$ formulas containing all subformulas of ϕ which is closed under negation (where double negations are removed; i.e., $\neg\neg\psi$ is identified with ψ), and such that $\psi_1 \,\mathcal{U}\, \psi_2 \in \mathsf{csub}(\phi)$ implies $\bigcirc(\psi_1 \,\mathcal{U}\, \psi_2) \in \mathsf{csub}(\phi)$. An *atom* (for ϕ) is a subset $\mathbf{a} \subseteq \mathsf{csub}(\phi)$ such that:

(1) for every $\psi \in \mathsf{csub}(\phi)$, $\{\psi, \neg\psi\} \cap \mathbf{a} \neq \emptyset$ and $\{\psi, \neg\psi\} \not\subseteq \mathbf{a}$;
(2) for every formula $\psi_1 \wedge \psi_2 \in \mathsf{csub}(\phi)$, $\psi_1 \wedge \psi_2 \in \mathbf{a}$ iff $\{\psi_1, \psi_2\} \subseteq \mathbf{a}$ and
(3) for all $\psi_1 \,\mathcal{U}\, \psi_2 \in \mathsf{csub}(\phi)$, $\psi_1 \,\mathcal{U}\, \psi_2 \in \mathbf{a}$ iff either $\psi_2 \in \mathbf{a}$ or $\{\psi_1, \bigcirc(\psi_1 \,\mathcal{U}\, \psi_2)\} \subseteq \mathbf{a}$.

In words, an atom is a maximally consistent subset of $\mathsf{csub}(\phi)$ which also verifies the satisfiability of the conjunction and the until operator locally. For the latter note that Condition (iii) states that satisfying $\psi_1 \,\mathcal{U}\, \psi_2$ is equivalent to satisfying ψ_2 now, or observing ψ_1 now and delaying the satisfaction of the properties of the until formula to the next point in time. From now on, the set of all atoms for ϕ is denoted by $\mathsf{At}(\phi)$. For brevity, we henceforth equate $\neg \circledcirc_{\bowtie p} \equiv \circledcirc_{\bowtie^- p}$, where $\bowtie^-$ is the inverse relation of $\bowtie$ and assume that formulas with $\circledcirc$ as main constructor are never negated in csub; e.g., $\neg \circledcirc_{\leq 0.5}\, a$ is replaced by $\circledcirc_{>0.5} a$.

An atom describes a set of formulas that can be satisfied in a node of a PLTL$_f$ interpretation, restricted only to those formulas that are relevant for identifying a model of ϕ; that is, $\mathsf{csub}(\phi)$. The conditions of maximality and local logical consistency guarantee that indeed, the sets of formulas satisfied at each node are atoms. To construct a model it thus suffices to build an atom-labelled tree which also preserves the semantics of the $\bigcirc$ and the $\circledcirc$ operators when traversing from a node to its children.

Recall that a run of a tree automaton $\mathcal{A}$ is a tree labelled with states of $\mathcal{A}$, and that we want to construct trees labelled with atoms of a formula ϕ. We thus construct a tree automaton which uses, as set of states, $\mathsf{At}(\phi)$. To define the transitions, we identify the combinations of probabilistic subformulas that can appear together under the uncertainty constraints. For example, if an atom contains $\circledcirc_{\leq 0.3}\psi_1$ and $\circledcirc_{\leq 0.4}\psi_2$, then transitions must contain successors with $\neg\psi_1$ and $\neg\psi_2$ (possibly combined into a single branch) as these formulas have a positive probability of being observed.

Given an atom $\mathbf{a} \in \mathsf{At}(\phi)$, let $\mathcal{P}(\mathbf{a}) = \{\circledcirc_{\bowtie p}\psi \in \mathbf{a}\}$ be the set of all probabilistic formulas appearing in $\mathbf{a}$. For every non-empty subset $S \subseteq 2^{\mathcal{P}(\mathbf{a})}$ define the system of inequalities

$$\mathfrak{I}(S) := \left\{ \sum_{\circledcirc_{\bowtie p_i}\psi_i \in Q, Q \in S} x_Q \bowtie p_i \mid \circledcirc_{\bowtie p_i}\psi_i \in \mathcal{P}(\mathbf{a}) \right\} \cup \left\{ x_Q \geq 0 \mid Q \in S \right\} \cup \left\{ \sum_{Q \in S} x_Q = 1 \right\}.$$

$\mathcal{S}(\mathbf{a})$ is the set of all $S \subseteq 2^{\mathcal{P}(\mathbf{a})}$ where $\mathfrak{I}(S)$ has a solution.[5]

Example 3.5. One possible atom of the formula ϕ_0 from Example 3.3 is

$$\mathbf{a}_0 = \left\{ \phi_0,\ \circledcirc_{\leq 0.5}\bigcirc a,\ \circledcirc_{\geq 0.6}(a\,\mathcal{U}\,b),\ \neg\bigcirc a,\ \neg a,\ \neg b,\ \neg(a\,\mathcal{U}\,b),\ \neg\bigcirc(a\,\mathcal{U}\,b) \right\}.$$

Then $\mathcal{P}(\mathbf{a}_0) = \{\circledcirc_{\leq 0.5}\bigcirc a, \circledcirc_{\geq 0.6}(a\,\mathcal{U}\,b)\}$. For brevity, let us call the elements of $\mathcal{P}(\mathbf{a}_0)$ 1 and 2, respectively (that is, 1 stands for the formula $\circledcirc_{\leq 0.5}\bigcirc a$ and 2 stands for $\circledcirc_{\geq 0.6}(a\,\mathcal{U}\,b)$). Each nonempty subset of $2^{\mathcal{P}(\mathbf{a}_0)} = \{\emptyset, \{1\}, \{2\}, \{1,2\}\}$ defines a system of inequalities.

The system of inequalities for $S_0 = \{\{1\}, \{2\}, \{1,2\}\}$

$$\begin{aligned} x_{\{1\}} + x_{\{1,2\}} &\leq 0.5 \\ x_{\{2\}} + x_{\{1,2\}} &\geq 0.6 \\ x_Q &\geq 0 \qquad\qquad Q \in S_0 \\ x_{\{1\}} + x_{\{2\}} + x_{\{1,2\}} &= 1, \end{aligned}$$

has a solution; e.g., $x_{\{1\}} = 0.4$, $x_{\{2\}} = 0.5$, $x_{\{1,2\}} = 0.1$. As argued later, this means that from the atom $\mathbf{a}_0$, it is possible to branch in three scenarios to satisfy the probabilities (see Figure 1(c)).

[5]Importantly, $S \subseteq 2^{\mathcal{P}(\mathbf{a})}$ is a set *of sets* of probabilistic formulas.

If we consider instead the set $S_1 = \{\emptyset, \{1\}, \{1,2\}\}$, the system $\mathfrak{I}(S_1)$

$$\begin{aligned} x_{\{1\}} + x_{\{1,2\}} &\leq 0.5 \\ x_{\{1,2\}} &\geq 0.6 \\ x_{\emptyset} &\geq 0 \\ x_{\{1\}} &\geq 0 \\ x_{\{1,2\}} &\geq 0 \\ x_{\emptyset} + x_{\{1\}} + x_{\{1,2\}} &= 1 \end{aligned}$$

has no solution: $x_{\{1,2\}}$ needs to be ≤ 0.5 and ≥ 0.6 simultaneously. Hence $S_0 \in \mathcal{S}(\mathbf{a}_0)$ but $S_1 \notin \mathcal{S}(\mathbf{a}_0)$. The latter means that it is impossible to satisfy $\mathbf{a}_0$ by branching into three different scenarios where in all of them, whenever $\bigcirc a$ is satisfied then $a \,\mathcal{U}\, b$ is also satisfied. It should be clear that this is the case since such a scenario contradicts the condition that the first formula has probability ≤ 0.5 while the latter one has probability ≥ 0.6.

Recall that $\mathcal{P}(\mathbf{a})$ contains all the probabilistic formulas appearing in the atom $\mathbf{a}$. Hence, if the original formula φ contains n probabilistic subformulas, the cardinality of $\mathcal{P}(\mathbf{a})$ is bounded by n and, moreover, each subset $S \subseteq 2^{\mathcal{P}(\mathbf{a})}$ has at most 2^n elements. We can enumerate all these subsets using only exponential space, through e.g., a recursive inclusion of elements. The system $\mathcal{I}(S)$ is a linear program with a number of constraints and variables which are polynomial on the cardinality of S—and hence exponential on n. It is well known that linear programs are solvable in polynomial time on the number of constraints and variables [25, 26]. Thus, overall constructing the sets $\mathcal{S}(\mathbf{a})$ requires exponential space on the number n of probabilistic formulas appearing in φ. On the other hand, this bound is independent of any other syntactic features of φ.

If $\mathcal{P}(\mathbf{a}) = \emptyset$, then $2^{\mathcal{P}(\mathbf{a})} = \{\emptyset\}$, and $\mathfrak{I}(\{\emptyset\}) = \{1{=}x_\emptyset\}$, which has a trivial solution. In other words, if an atom $\mathbf{a}$ contains no probabilistic subformulas, $\mathcal{S}(\mathbf{a})$ contains only one element, and the construction reduces to the classical scenario of LTL_f. It is thus helpful to think of our automata construction as a generalisation of the known approach for LTL_f [13, 46]. Briefly, the automata-based method for deciding satisfiability of LTL_f formulas uses a string automaton whose states are formed by all maximally consistent subsets of "relevant" formulas which can be observed at different points in time in a trace—thus representing the status of an execution at each point in time—while the transitions ensure that the temporal constraints are satisfied at successive points in time; see Definition 3.7 for details.

Each element in $\mathcal{S}(\mathbf{a})$ defines a set of tuples of atoms yielding the transition relation of the automaton. Assume w.l.o.g. that the elements of each $S \in \mathcal{S}(\mathbf{a})$ are ordered as $Q_1, \ldots, Q_{|S|}$. $T_S(\mathbf{a})$ is the set of $|S|$-tuples of atoms $(\mathbf{a}_1, \ldots, \mathbf{a}_{|S|})$ such that:

(i) for all $\bigcirc\psi \in \mathsf{csub}(\phi)$, $\bigcirc\psi \in \mathbf{a}$ iff $\psi \in \mathbf{a}_i$ for all i, and
(ii) for all $\circledcirc_{\bowtie p}\psi \in \mathsf{csub}(\phi)$ and every $i, 1{\leq}i{\leq}|S|$, $\circledcirc_{\bowtie p}\psi \in Q_i$ iff $\psi \in \mathbf{a}_i$.

Example 3.6. From Example 3.5, $S_0 \in \mathcal{S}(\mathbf{a})$ defines 3-tuples of atoms such that the first two elements contain one of the probabilistic subformulas each, and the last contains both probabilistic subformulas. Hence, the tuple $(\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3)$ formed by the following atoms belongs to $T_{S_0}(\mathbf{a})$:[6]

$$\begin{aligned} \mathbf{a}_1 &= \{\ \neg\phi_0,\ \circledcirc_{>0.5} \bigcirc a,\ \circledcirc_{<0.6}(a \,\mathcal{U}\, b),\ \bigcirc a,\ \neg a,\ \neg b,\ \neg(a \,\mathcal{U}\, b),\ \bigcirc(a \,\mathcal{U}\, b)\ \} \\ \mathbf{a}_2 &= \{\ \neg\phi_0,\ \circledcirc_{>0.5} \bigcirc a,\ \circledcirc_{<0.6}(a \,\mathcal{U}\, b),\ \neg\bigcirc a,\ a,\ \neg b,\ a \,\mathcal{U}\, b,\ \bigcirc(a \,\mathcal{U}\, b)\ \} \\ \mathbf{a}_3 &= \{\ \neg\phi_0,\ \circledcirc_{>0.5} \bigcirc a,\ \circledcirc_{<0.6}(a \,\mathcal{U}\, b),\ \bigcirc a,\ \neg a,\ b,\ a \,\mathcal{U}\, b,\ \neg\bigcirc(a \,\mathcal{U}\, b)\ \} \end{aligned}$$

[6]Recall that we equate $\neg \circledcirc_{\leq 0.5}\, \phi \equiv \circledcirc_{>0.5}\phi$.

Note that there are many other triples in $T_{S_0}(\mathbf{a})$; the important aspect is that the elements in the triple are all atoms, that the first atom contains $\bigcirc a$, the second contains $a \,\mathcal{U}\, b$, and the third contains both formulas. Moreover, there should be at least one atom that includes $\neg a$—since otherwise we would need $\mathbf{a}_0$ to contain $\bigcirc a$ and similarly, at least one atom contains $\neg(a \,\mathcal{U}\, b)$.

We use these tuples to define the transitions of the automaton which verifies satisfiability of PLTL$_f$ formulas. Specifically, the tuples describe the branching in the tree model that a successful run of the automaton constructs. This idea is formalised next.

Definition 3.7. Let ϕ be a PLTL$_f$ formula. The tree automaton $\mathcal{A}_\phi = (Q, \Delta, I, F)$ is defined by:

- $Q = \mathsf{At}(\phi)$;
- $\Delta = \{\{\mathbf{a}\} \times \bigcup_{S \in \mathcal{S}(\mathbf{a})} T_S(\mathbf{a}) \mid \mathbf{a} \in Q\}$;
- $I = \{\mathbf{a} \in Q \mid \varphi \in \mathbf{a}\}$ and
- F is the set of all atoms *not* containing any formula of the form $\bigcirc\psi$, $\circledcirc_{>p}\psi$, or $\circledcirc_{\geq p'}\psi$ with $p' > 0$.

The construction of $\mathcal{A}_\phi$ naturally generalises the automata-based approach for satisfiability of LTL$_f$ formulas: if ϕ has no probabilistic constructor (i.e., it is an LTL$_f$ formula), $\mathcal{A}_\phi$ is the standard automaton for this setting [10]. $\mathcal{A}_\phi$ accepts a class of tree-shaped structures which almost fully describe a model of the formula ϕ, except for the probabilities assigned to each branch, and where redundant branches are merged. Hence, the only missing element to have a model are the probabilistic values attached to each successor of a node. These are found solving the system of inequalities built from each transition. Importantly, any solution of the system provides adequate probabilities. Hence, the fact that a solution exists is a sufficient condition to guarantee the existence of a model.

Theorem 3.8. *The PLTL$_f$ formula ϕ is satisfiable iff $\mathcal{L}(\mathcal{A}_\phi) \neq \emptyset$.*

Proof. [$\Leftarrow$] Suppose first that $\mathcal{L}(\mathcal{A}) \neq \emptyset$, and consider a tree $T \in \mathcal{L}(\mathcal{A})$. Since T is accepted by $\mathcal{A}$, there exists a successful run ρ of $\mathcal{A}$ over T, which labels every node $w \in T$ with an atom $\rho(w)$. Let A be the set of all propositional variables appearing in ϕ. Define the function $\cdot^I : T \to 2^A$ by $w^I := \rho(w) \cap A$. Moreover, for every node w with k successors, as ρ is a successful run, it holds that $(\rho(w1), \ldots, \rho(wk)) \in T_S$ for some $S \in \mathcal{S}(\rho(w))$. The latter means that the system $\mathfrak{I}(S)$ has a solution for the (ordered) variables $x_1, \ldots, x_k$ in $[0, 1]$. Hence, we define the function $P : T \setminus \{\varepsilon\} \to [0, 1]$ where $P(w\ell)$ is the solution of the system in $\rho(w)$ for the variable x_ℓ. Overall, this defines an interpretation $I = (T, \cdot^I, P)$. We show, by induction on the structure of the formulas, that for every $\psi \in \mathsf{csub}(\phi)$ and every $w \in T$, it holds that $I, w \models \psi$ iff $\psi \in \rho(w)$. In particular, since ρ is such that $\phi \in \rho(\varepsilon)$, this implies that I is a model of ϕ.

For a propositional variable $a \in A$ the result holds trivially by construction, so we focus on the remaining constructors. Assume, as an induction hypothesis, that the result holds for every formula in $\mathsf{csub}(\psi) \cup \mathsf{csub}(\psi_1) \cup \mathsf{csub}(\psi_2)$.

[$\neg$] $I, w \models \neg\psi$ iff $I, w \not\models \psi$ iff (induction hypothesis) $\psi \notin \rho(w)$ iff (atom maximality) $\neg\psi \in \rho(w)$.

[$\wedge$] $I, w \models \psi_1 \wedge \psi_2$ iff $I, w \models \psi_1$ and $I, W \models \psi_2$ iff (induction hypothesis) $\{\psi_1, \psi_2\} \subseteq \rho(w)$ iff (atom condition (ii)) $\psi_1 \wedge \psi_2 \in \rho(w)$.

[$\bigcirc$] $I, w \models \bigcirc\psi$ iff w is not a leaf and for every $wi \in T$ $I, wi \models \psi$ iff for every $wi \in T$, $\psi \in \rho(wi)$ iff (definition of the transition relation) $\bigcirc\psi \in \rho(w)$.

[$\mathcal{U}$] $I, w \models \psi_1 \,\mathcal{U}\, \psi_2$ iff (i) $I, w \models \psi_2$ or (ii) $I, w \models \psi_1$, w is not a leaf, and for all $wi \in T$, $I, wi \models \psi_1 \,\mathcal{U}\, \psi_2$. We show that $\psi_1 \,\mathcal{U}\, \psi_2 \in \rho(w)$ by induction on the subtree rooted at w. If w is a leaf node, only case (i) is possible, and so $I, w \models \psi_1 \,\mathcal{U}\, \psi_2$ iff $\psi_2 \in \rho(w)$ which means that $\psi_1 \,\mathcal{U}\, \psi_2 \in \rho(w)$ by the definition of an atom. If w is not a leaf node. Case (i) is treated as for the leaf nodes. Case (ii) holds

iff $\psi_1 \in \rho(w)$ and, by the second induction, for every $wi \in T$, $\psi_1 \mathcal{U} \psi_2 \in \rho(wi)$, which implies by the definition of the transition relation that $\bigcirc(\psi_1 \mathcal{U} \psi_2) \in \rho(w)$, and since $\rho(w)$ is an atom, $\psi_1 \mathcal{U} \psi_2 \in \rho(w)$.
[$\circledcirc$] $I, w \models \circledcirc_{\bowtie p}\psi$ iff $\sum_{wi \in T, I, wi \models \psi} P(wi) \bowtie p$. By construction and the induction hypothesis, the latter holds iff $\sum_{wi \in T, \psi \in \rho(wi)} x_i \bowtie p$, where the x_is for the solution of the system in $\rho(w)$. This is only possible if $\circledcirc_{\bowtie p}\psi \in Q_i \subseteq \rho(w)$.

Hence, I is a model of ϕ and ϕ is satisfiable. This finishes this direction of the proof.
[$\Rightarrow$] Conversely, suppose that ϕ is satisfiable, and let $I = (T, \cdot^I, P)$ be a model of ϕ. Recall that in PLTL$_f$, a model is a labelled tree. We will use this tree to construct a successful run of $\mathcal{A}$, but first need to adapt it to a simplified form.

Given a node $w \in T$, let $\mathcal{P}(w) \subseteq \mathsf{csub}(\phi)$ be the set of all probabilistic formulas $\circledcirc_{\bowtie p}\psi \in \mathsf{csub}(\phi)$ such that $I, w \models \circledcirc_{\bowtie p}\psi$, and define $S(w) \subseteq 2^{\mathcal{P}(w)}$ to be the set of subsets $O \subseteq \mathcal{P}(w)$ such that there is a successor $wi \in T$ that satisfies $I, wi \models \psi$ for all $\circledcirc_{\bowtie p}\psi \in O$ and $I, wi \not\models \psi$ for all $\circledcirc_{\bowtie p}\psi \notin O$. Given an $O \in S(w)$, if there are two successors $wi, wj \in T$ that satisfy the previous conditions, it is possible to *prune* the tree T by removing all nodes of the form $wjv, v \in \mathbb{N}^*$, and setting $P'(wi) := P(wi) + P(wj)$. Afterwards, we may need to rename some of the nodes to guarantee that T is closed under preceding siblings.

It is easy to see that $I' = (T', \cdot^{I'}, P')$, where T' is the prunned tree and $\cdot^{I'}$ is $\cdot^I$ restricted to T' is also a model of ϕ. Let $I_0 = (T_0, \cdot^{I_0}, P_0)$ be the result of applying this prunning procedure to all nodes in the original model (in a top-down manner). Then, it is a simple exercise to verify that the labelling $\rho : T_0 \to \mathsf{At}(\phi)$ where

$$\rho(w) = \{\psi \in \mathsf{csub}(\phi) \mid I_0, w \models \psi\}$$

is in fact a successful run of $\mathcal{A}$, and hence $\mathcal{L}(\mathcal{A}) \neq \emptyset$. □

Automata emptiness is decidable in time $O(|Q|^{k+2})$, where k is the rank of the automaton. In this case, the rank of $\mathcal{A}_\phi$ depends on the size of the formula ϕ; specifically, on the number of probabilistic subformulas that it contains: if $\mathsf{csub}(\phi)$ has n probabilistic subformulas, the rank of $\mathcal{A}_\phi$ is bounded by 2^n; i.e., emptiness of $\mathcal{A}_\phi$ runs in time $O(|Q|^{2^n})$. There is also a non-deterministic algorithm that uses space $O(|Q| \cdot 2^n)$. As the number of states is bounded exponentially on the length of ϕ, and computing them requires only exponential space on n (see the discussion after Example 3.5), Savitch's theorem [41] yields the following result.

Theorem 3.9. *PLTL$_f$ satisfiability is decidable in exponential space in the number of probabilistic formulas, but only exponential time in the size of the formula (not counting the number of probabilistic constructors).*

One of the important consequences of the second part of this theorem is that, if the total number of probabilistic formulas is bounded by some constant, or if we parameterise the problem over the number of probabilistic subformulas [16], then satisfiability of PLTL$_f$ formulas is in ExpTime. On the other hand, this bound cannot be further improved because satisfiability is ExpTime-hard on the length of the input formula ϕ. The proof of this fact is based on a reduction from the *intersection non-emptiness* problem for deterministic tree automata [8, 43].

Theorem 3.10. *PLTL$_f$ satisfiability is ExpTime-hard.*

Proof. We prove ExpTime-hardness by a reduction from the intersection non-emptiness problem for deterministic automata over labelled trees. A *deterministic tree automaton over labelled trees* is a tuple $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ where Q, I, and F are as in Definition 2.1, Σ is a finite called the *alphabet*, and $\Delta : Q \times \Sigma \to \bigcup_{i \leq k} Q^k$ is a total *transition function*. Given a Σ-labelled tree, a run of

$$\psi_{q,\sigma} := \begin{cases} q \wedge \sigma \rightarrow \left(x_{\mathsf{END}} \vee \bigwedge_{i=1}^{k_{q,\sigma}} \circledcirc_{\geq 1/k_{q,\sigma}} (q_i^{q,\sigma} \wedge i)\right) & q \in F \\ q \wedge \sigma \rightarrow \left(\neg x_{\mathsf{END}} \wedge \bigwedge_{i=1}^{k_{q,\sigma}} \circledcirc_{\geq 1/k_{q,\sigma}} (q_i^{q,\sigma} \wedge i)\right) & q \notin F \end{cases}$$

$$\psi_Q := \bigvee_{q \in Q} \left(q \wedge \bigwedge_{q' \in Q \setminus \{q\}} \neg q' \right)$$

$$\psi_\Sigma := \bigvee_{\sigma \in \Sigma} \left(\sigma \wedge \bigwedge_{\sigma' \in \Sigma \setminus \{\sigma\}} \neg \sigma' \right)$$

$$\psi_N := \bigwedge_{i=1}^{k} \left(i \rightarrow \bigwedge_{j \neq i} \neg j \right)$$

Fig. 3. Formulas describing the automaton for the proof of Theorem3.10.

this automaton is a Q-labelling that is consistent with the transition function. All the associated notions are defined in the obvious way. The intersection non-emptiness problem for these automata consists in deciding, given n such automata $\mathcal{A}_i, 1 \leq i \leq n$ with disjoint sets of states, whether $\bigcap_{i=1}^{n} \mathcal{L}(\mathcal{A}_i) \neq \emptyset$. This problem is known to be ExpTime-hard [43]. We first construct a formula for each automaton, and then use the conjunction of these formulas to deal with the intersection of automata.

Given a deterministic automaton $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$, we build the PLTL$_f$ formula $\varphi_{\mathcal{A}}$ as follows. The propositional variables appearing in the formula will be given by the elements of $Q \cup \Sigma \cup \{1, \ldots, k\}$; that is, the states, the symbols and the first k natural numbers. Intuitively, an interpretation and node mapping $q \in Q$ to means that the state q holds in that element, and analogously for $\sigma \in \Sigma$. The variables $1, \ldots, k$ are used to distinguish the different successors of a node in a tree. This intuition will become more clear after the construction of the formula. In addition, we have a new propositional variable x_{END} that identifies the leaf nodes.

For every $(q, \sigma) \in Q \times \Sigma$, let $\Delta(q, \sigma) = (q_1^{q,\sigma}, \ldots, q_{k_{q,\sigma}}^{q,\sigma})$ and define the formulas $\psi_{q,\sigma}$, ψ_Q, ψ_Σ and ψ_N as in Figure 3.

Then, we set

$$\varphi_{\mathcal{A}} := \bigvee_{q \in I} q \wedge \Box \left(\psi_Q \wedge \psi_\Sigma \wedge \bigwedge_{(q,\sigma) \in Q \times \Sigma} \psi_{q,\sigma} \right).$$

Note that the length of $\varphi_{\mathcal{A}}$ is polynomially bounded by the size of $\mathcal{A}$. We first show that every model of $\varphi_{\mathcal{A}}$ can be transformed into a tree accepted by $\mathcal{A}$, and conversely, every tree accepted by $\mathcal{A}$, together with a successful run, is a representation of a model of $\varphi_{\mathcal{A}}$.

Let $J = (T, \cdot^J, P)$ be a model of φ. By construction, (see formulas ψ_Q and ψ_Σ), for every node $w \in T$ there is exactly one $\sigma \in \Sigma$ and one $q \in Q$ such that $\sigma, q \in w^J$. Abusing the notation, we call these elements $\Sigma(w)$ and $Q(w)$, respectively. We can assume, w.l.o.g., that for every non-leaf node $w \in T$, if $q, \sigma \in w^J$, then for every $j, 1 \leq j \leq k_{q,\sigma}$, $wj \in T$ and $j \in wj^J$.[7] We construct the labelled tree $T_J : T \rightarrow \Sigma$ where $T_J(w) = \Sigma(w)$ for all $w \in T$. We show that $T_J \in \mathcal{L}(\mathcal{A})$, by using the function $Q : T \rightarrow Q$ to build a successful run of $\mathcal{A}$ on this tree. Note that for the root node ε, $Q(\varepsilon) \in I$, because J is a model of $\bigvee_{q \in I} q$. For every leaf node $w \in T$, $Q(w) \in F$ because otherwise the formula $\psi_{q,\sigma}$ guarantees that w must have a successor node. Finally, given a non-leaf node $w \in T$, let $q = Q(w)$ and $\sigma = \Sigma(w)$ and $\Delta(q, \sigma) = (q_1^{q,\sigma}, \ldots, q_{k_{q,\sigma}}^{q,\sigma})$. Since J is a model of $\psi_{q,\sigma}$, and

[7] Otherwise, one can merge different successors $w\ell$ such that $j \in w\ell^J$ and reorder the successors to satisfy the condition.

by the assumption stated before, w must have $k_{q,\sigma}$ successors, and is such that $Q(wj) = q_j^{q,\sigma}$ for all $j, 1 \leq j \leq k_{q,\sigma}$. Thus, the labelling of T provided by the function Q forms a successful run, and $T_J \in \mathcal{L}(\mathcal{A})$.

For the second claim, let T be a Σ-labelled tree, and $Q : T \to \boldsymbol{Q}$ a successful run of $\mathcal{A}$ over T. We build a model $J = (T, \cdot^J, P)$ of $\varphi_{\mathcal{A}}$ as follows. For every $w \in T$, $\sigma \in \Sigma$, and $q \in \boldsymbol{Q}$, we have $\sigma \in w^J$ iff $T(w) = \sigma$ and $q \in w^J$ iff $Q(w) = q$. In addition, for every node $wj \in T$, $j \in wj^J$, and for every leaf node w $x_{\mathsf{END}} \in w^J$. Finally, if w has ℓ successors, then for every $i, 1 \leq i \leq \ell$, $P(wi) = 1/\ell$. It is easy to see that, since Q is a successful run of $\mathcal{A}$ over T, this interpretation satisfies all the constraints of $\varphi_{\mathcal{A}}$, and hence it is a model of this formula.

Given n deterministic tree automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$, we construct the PLTL$_f$ formula $\varphi := \bigwedge_{i=1}^n \varphi_{\mathcal{A}_i}$, whose length is polynomially bounded by the total length of the n automata. We show, making use of the previous arguments, that this formula is satisfiable iff the intersection of these automata is not empty.

If there is a tree T in the intersection of these automata, then there is a successful run for $\mathcal{A}_i$ over T for all $i, 1 \leq i \leq n$. We can use these successful runs to build a model of φ as did in the previous paragraph. Conversely, let J be a model of φ. In particular, $J = (T, \cdot^J, P)$ is a model of $\varphi_{\mathcal{A}_i}$ for all $i, 1 \leq i \leq n$. Thus, as we have shown already, the labelled tree T_J (which only depends on the alphabet symbols Σ) is in $\mathcal{L}(\mathcal{A}_i)$ for all i; i.e., $T_J \in \bigcap_{i=1}^n \mathcal{L}(\mathcal{A}_i)$, and hence the intersection is not empty. □

This theorem shows that deciding whether a PLTL$_f$ is satisfiable is a hard problem in terms of computational complexity. In Section 5 we will introduce a fragment of this logic where this complexity is reduced. But before that we study a different kind of reasoning problem.

4 Probabilistic Entailment

In the previous section, we focused on a decision problem considering only the existence of a model of the PLTL$_f$ formula ϕ, disregarding the probabilistic information included in this formula ϕ. We now consider reasoning problems dealing with the likelihood of different traces. A basic probabilistic reasoning problem on PLTL$_f$ is computing the *most likely* trace—that is, the trace that would be more easily observed—along with its probability. To handle the multiplicity of models (we cannot know which of the many available models is in fact being observed), we use an optimistic approach, which selects the model maximising the likelihood of observing a given trace. One could alternatively choose a *pessimistic* approach minimising the likelihood instead. Such a case can be handled analogously by changing all relevant maxima for minima in the following. To avoid tedious repetitions, we disregard such a pessimistic approach in this work.

Definition 4.1 (Trace Probability). A *trace* is a finite sequence of propositional valuations. We say that the interpretation $I = (T, \cdot^I, P)$ *contains* the trace t if there is a branch b of T such that $\mathsf{b}^I = t$. The *probability* of the trace t in I is $P_I(t) = \max_{\mathsf{b}^I = t} \prod_{w \in \mathsf{b}} P(w)$. The *probability* of t w.r.t. the PLTL$_f$ formula ϕ is $P_\phi(t) = \max_{I \models \phi} P_I(t)$.

In words, the probability of a trace t in an interpretation I is the maximum probability that is assigned by a branch in I that equals t, and the probability of a trace in the formula ϕ is the highest probability that can be assigned in any model of ϕ (and hence, in any branch appearing in any model of ϕ).

Example 4.2. Using the formula ϕ_0 from Example 3.3, let I_0, I_1 and I_2 be the models (a), (b) and (c) from Figure 1, respectively. I_2 contains the trace $(\emptyset, \{b\}, \{a\})$, but the other two models do not. The models I_0 and I_2 contain the trace $t_0 = (\emptyset, \{b\}, \emptyset, \{a\})$. In this case, $P_{I_0}(t_0) = 1$; and $P_{I_2}(t) = 0.5$.

In particular, we get that $P_{\phi_0}(t_0) = 1$ as witnessed by the model I_0. On the other hand, the trace $t_1 = (\emptyset, \{a\}, \{a\}, \{b\})$ has probability 0.5; see model I_1.

We are interested in detecting the traces that maximise this probability, as formalised in the next definition.

Definition 4.3 (mlt). The trace t is a *most likely trace* (mlt) w.r.t. ϕ iff for every trace t', it holds that $P_\phi(t') \leq P_\phi(t)$.

In our running example, a most likely trace is $(\emptyset, \{b\}, \emptyset, \{a\})$, which has probability 1 (Figure 1(a)). However, mlts are not necessarily unique; in fact, they seldom are. Indeed, continuing with our example formula ϕ_0, one can see that $(\emptyset, \{b\}, \emptyset, \{a, b\})$ and $(\emptyset, \{a\}, \{b\}, \emptyset)$ are also mlts w.r.t. ϕ_0. More generally, a most likely trace is one that maximises its probability in all possible models of the formula. When we speak of a trace, it is already a full observation of the evolution of the system, which means that all the superpositions have been resolved and it thus suffices to verify the branches of the tree models directly.

The next question is how to find such traces automatically. More generally, how can one compute the probability of a given trace. To answer these questions, we take advantage of the theory of weighted automata. Interestingly, though, we can restrict our attention to *string* automata, abstracting from the specific tree-shape of the models, as described next.

4.1 Most Likely Traces

To find the mlts w.r.t. ϕ, we transform the tree automaton $\mathcal{A}_\phi$ into a weighted string automaton $\mathcal{B}_\phi$ which keeps track of the most likely transitions available from a given state of $\mathcal{A}_\phi$. For brevity, we do not distinguish between the valuation forming a model, and the atom (containing the valuation) of the run of the automaton. Using the probabilistic semiring $\mathbb{P}$, the behaviour of $\mathcal{B}_\phi$ yields the probability of the mlts. We later show how to use this information to extract the actual traces that reach this maximum probability.

Recall that the reduced automaton $\ddot{\mathcal{A}}_\phi$ of the emptiness test excludes the bad states from $\mathcal{A}_\phi$ and accepts the same language as $\mathcal{A}_\phi$, but from every state in $\ddot{\mathcal{A}}_\phi$ one can build a successful run (see Section 2.1). Deleting bad states also removes all transitions (produced by the different combinations of the probabilistic subformulas that appear in an atom) which cannot be used due to semantic incompatibilities. Hence, all the elements that appear in a transition can appear in a path in a model. More formally, let $(\mathbf{a}, \mathbf{a}_1, \ldots, \mathbf{a}_n) \in \ddot{\Delta}$; i.e., a transition from $\ddot{\mathcal{A}}_\phi$. There exists an $S \in \mathcal{S}(\mathbf{a})$ such that $(\mathbf{a}_1, \ldots, \mathbf{a}_n) \in T_S$; otherwise, it would not be a transition in $\mathcal{A}_\phi$, nor in $\ddot{\mathcal{A}}_\phi$. Now, for each $Q \in S$, we solve the optimisation problem:

$$\text{maximize } x_Q \qquad\qquad \text{subject to } \Im(S).$$

Intuitively, we compute the largest probability that a branch satisfying the probabilistic constraints in Q can obtain, given the other branches defined by S. Call the result of this optimisation problem $\mathsf{m}_{S,\mathbf{a}}(Q)$—this is the maximum value that a transition from $\mathbf{a}$ to a branch defined by Q can get, when using the set S. Importantly, as each optimisation problem is solved independently, the maxima may not add 1; e.g., for $\mathbf{a}_0, S_0$ from Example 3.5, $\mathsf{m}_{S_0,\mathbf{a}_0}(\{2\}) = 1$, $\mathsf{m}_{S_0,\mathbf{a}_0}(\{1,2\}) = 0.5$ and $\mathsf{m}_{S_0,\mathbf{a}_0}(\{1\}) = 0.4$. This is intended; we try to identify the *largest* probability that can be assigned to a path in a model; i.e., a trace. Of course, these largest probabilities will be achieved, for different branches, through different models, but that is not an issue for our reasoning problem (which maximises over all possible models), nor for our approach.

For an atom $\mathbf{a}$ and a set $Q \subseteq \mathcal{P}(\mathbf{a})$, let now

$$\mathsf{m}_\mathbf{a}(Q) := \max_{S \in \mathcal{S}(\mathbf{a}), Q \in S} \mathsf{m}_{S,\mathbf{a}}(Q).$$

We obtain the automaton $\mathcal{B}_\phi$ by *flattening* $\ddot{\mathcal{A}}_\phi$ into a string automaton, and weighting every transition according to the functions m just defined.

Definition 4.4 (Flattened Automata). Let ϕ be a PLTL$_f$ formula, and $\mathcal{A}_\phi$ the tree automaton constructed from it (Definition 3.7). $\mathcal{B}_\phi = (\ddot{Q}, \mathsf{in}, \mathsf{wt}, \ddot{F})$ is the weighted automaton where $\ddot{Q}$ and $\ddot{F}$ are the states and final states from the reduced automaton $\ddot{\mathcal{A}}_\phi$, and

$$\mathsf{in}(\mathbf{a}) = \begin{cases} 1 & \mathbf{a} \in \ddot{I} \\ 0 & \text{otherwise;} \end{cases} \qquad \mathsf{wt}(\mathbf{a}, \mathbf{a}') = \begin{cases} \max_{Q \in \mathbf{a}'} \mathsf{m}_{\mathbf{a}}(Q) & (\mathbf{a}, \ldots, \mathbf{a}', \ldots) \in \ddot{\Delta} \\ 0 & \text{otherwise.} \end{cases}$$

It is important to highlight that $\mathcal{B}_\phi$ is constructed from the reduced automaton $\ddot{\mathcal{A}}_\phi$ and not from the original $\mathcal{A}_\phi$. This ensures that branches defined by unsatisfiable constraints are ignored in the computation, as they will not appear in any model. Consider for example an atom $\mathbf{a}$ such that $\{\circledcirc_{\leq p} a, \circledcirc_{\leq q} \neg a\} \subseteq \mathbf{a}$ with $p + q < 1$. Then we get that $\mathcal{S}(\mathbf{a}) \neq \emptyset$ and $\mathsf{m}_{\mathbf{a}}(\{\circledcirc_{\leq p} a\}) = p$. However, if $\mathbf{a}$ is not a final state, then it must be a bad state: in fact, $\mathbf{a}$ has no transitions starting from it. Constructing $\mathcal{B}_\phi$ from $\mathcal{A}_\phi$, $\mathbf{a}$ would have a transition to an atom $\mathbf{a}'$ containing a (with weight p), which could yield an incorrect result.

In the following, we see that the flattened automaton allows us to compute the probability of the most likely traces with respect to the formula ϕ.

Theorem 4.5. *Let $\mathcal{B}_\phi$ be the weighted automaton constructed from the PLTL$_f$ formula ϕ through Definition 4.4. The probability of the mlt w.r.t. ϕ is $\|\mathcal{B}_\phi\|$.*

Proof. Notice first that the probability of the mlt is zero iff ϕ is unsatisfiable. In this case, the automata $\ddot{\mathcal{A}}_\Phi$ and $\mathcal{B}_\phi$ become empty, and hence the behaviour of the latter is zero as well. So we are only interested in cases where this probability is greater than 0.

Consider first a run $\rho = q_0, \ldots, q_n$ of $\mathcal{B}_\phi$ such that $\mathsf{wt}(\rho) > 0$. In particular this means that $\mathsf{wt}(q_i, q_{i+1}) > 0$ for all $i, 1 \leq i < n$. By construction, this means that for every $i, 1 \leq i < n$ there is a transition of $\ddot{\mathcal{A}}_\phi$ (i.e., a tuple $\delta \in \ddot{\Delta}$) of the form $(q_i, \ldots, q_{i+1}, \ldots, q_k)$ such that $\mathsf{wt}(q_i, q_{i+1})$ is the maximum value that can be given to a successor of a node satisfying q_i which satisfies q_{i+1}. Moreover, all states appearing in this transition are *good* states, which means that a successful run can still be constructed from them. Thus, there is a successful run of $\ddot{\mathcal{A}}_\phi$ (and hence of $\mathcal{A}_\phi$) which has ρ as a branch. If $\mathsf{in}(q_0) = 1$ (that is, if q_0 is an initial state of $\mathcal{A}_\phi$), as in the proof of Theorem 3.8, we can build a model of ϕ containing a branch $t = \rho(q_0) \cap A, \ldots, \rho(q_n) \cap A$, where A is the set of all propositional variables in ϕ. Moreover, the probability of this trace in this model is exactly $\mathsf{wt}(\rho)$. Thus, to summarise, for every successful run ρ of $\mathcal{B}_\phi$, there is a model I and a trace t such that $\mathsf{wt}(\rho) = P_I(t)$. This implies that the probability of the mlt is greater or equal to $\|\mathcal{B}_\phi\|$.

Conversely, consider a model I containing the trace t. As in the proof of Theorem 3.8, we can assume without loss of generality that this model translates into a successful run ρ of $\ddot{\mathcal{A}}_\phi$. In this model, for every non-root node wi, it holds that $P(wi) \leq \mathsf{m}_{S,\rho(w)}(Q)$, where S and Q are the ones obtained from the transition used in ρ. In particular, $P(wi) \leq \mathsf{m}_{\rho(w)}(Q)$, and hence $P_I(t) \leq \|\mathcal{B}_\phi\|$. Since this is true for all traces and all models, it follows that the probability of the most likely trace is at most $\|\mathcal{B}_\phi\|$.

Piecing both parts together yields the desired result. □

The behaviour of $\mathcal{B}_\phi$ is computable in polynomial time on the number of states, i.e., exponential on the length of the formula, but is not affected by the number of probabilistic subformulas appearing in ϕ. Yet, to build $\mathcal{B}_\phi$, we need first to construct and manipulate $\mathcal{A}_\phi$, which may be doubly-exponential on the number of probabilistic subformulas. Indeed, there is a trace with positive probability iff ϕ is satisfiable. Thus, deciding whether the probability of the most likely trace is higher than some bound has the same complexity as satisfiability.

Corollary 4.6. *The probability of the mlts is computable in exponential space in the number of probabilistic formulas, but exponential time in the size of the formula. Deciding if it is greater than 0 is ExpTime-hard.*

So far we have seen how to compute the probability of the mlts. Yet, in applications related to process modelling, one is usually not interested in the probabilities themselves, but rather on the *traces* associated to that probability. To find the mlts (and not just their probability), we adapt the computation process for the behaviour of $\mathcal{B}_\phi$. At each iteration of the computation, associate each state with the maximum probability that can be derived from a trace starting from it. Together with this number, we also store the successor states yielding that maximum probability. This yields an (unweighted, string) automaton that accepts all—and only—the most likely traces.

Definition 4.7. Given a PLTL$_f$ formula ϕ, its weighted automaton $\mathcal{B}_\phi = (Q, \mathsf{in}, \mathsf{wt}, F)$, and a state $\mathbf{a} \in Q$, let $\mathsf{w}(\mathbf{a})$ be obtained through the computation of the behaviour of $\mathcal{B}_\phi$. The unweighted automaton $\overline{\mathcal{B}_\phi} = (Q, I, \Delta, F)$ is given by

$$I = \{\mathbf{a} \in Q \mid \mathsf{in}(\mathbf{a}) \cdot \mathsf{w}(\mathbf{a}) = \|\mathcal{B}_\phi\|\},$$
$$\Delta = \{(\mathbf{a}, \mathbf{a}') \in Q \times Q \mid \mathsf{wt}(\mathbf{a}, \mathbf{a}') \cdot \mathsf{w}(\mathbf{a}') = \mathsf{w}(\mathbf{a})\}.$$

As explained before, this automaton only preserves the transitions that allow for the highest probabilities in traces, and thus only those that generate most likely traces. Special attention should be paid to the case of final states. Note that in $\mathcal{B}_\phi$ there may exist transitions from a final state to other states, which allow for longer traces. If those additional transitions reduce the overall probability of the trace (because at least one has weight smaller than 1), they are ignored in the behaviour computation. We simulate this behaviour in the transition relation Δ of $\overline{\mathcal{B}_\phi}$: a final state can only have an outgoing transition if it is possible to continue the trace using only transitions of weight 1.

Theorem 4.8. *The unweighted $\overline{\mathcal{B}_\phi}$ accepts exactly the set of most likely traces.*

Proof. By construction, the initial states of $\overline{\mathcal{B}_\phi}$ are exactly those that maximise the likelihood of the trace, and likewise transitions are only allowed when they preserve the maximum possible probability. That is, for every successful run ρ of $\overline{\mathcal{B}_\phi}$, if seen over the weighted automaton $\mathcal{B}_\phi$ we get that $\mathsf{wt}(\rho) = \|\mathcal{B}_\phi\|$. Following the arguments from the proof of Theorem 4.5, such a run corresponds to a trace t in a model I such that $P_I(t) = \|\mathcal{B}_\phi\|$, but since the latter is the probability of the most likely traces, t must be an mlt as well. □

As a small corollary of this theorem, we obtain that the class of most likely traces for a PLTL$_f$ formula forms a regular language. Hence, this class can be described compactly through a regular expression or other similar means.

4.2 Trace Probability

While it is important to find and understand the most likely traces and their properties, it is often more useful to compute the likelihood of observing a specific trace or an element from a set of traces. For instance, we may want to verify that an unwanted outcome is unlikely, if not impossible. This problem can be reduced to that of computing the most likely traces, as long as the set of traces of interest is also a regular language. In the following, we call a set of finite traces L *recognisable* iff it is a regular language. Note in particular that every finite set of traces is recognisable [24].

Definition 4.9 (Language Probability). Let L be a recognisable set of finite traces and ϕ a PLTL$_f$ formula. The *probability* of L w.r.t. ϕ is defined by

$$P_\phi(L) = \max_{t \in L} P_\phi(t).$$

A trace $t \in L$ is a *most likely trace of* L w.r.t. ϕ iff it holds that $P_\phi(t) = P_\phi(L)$.

Since L is recognisable, there exists an automaton $\mathcal{A}_L$ that accepts exactly the set of traces belonging to L. We can obviously see this automaton as a very simple weighted automaton, whose weights are all in $\{0, 1\}$: we simply represent I and Δ through their characteristic functions. To find the mlts of L and their corresponding probability, we intersect $\mathcal{A}_L$ with $\mathcal{B}_\phi$ and $\overline{\mathcal{B}_\phi}$, respectively, and compute $\|\mathcal{A}_L \cap \mathcal{B}_\phi\|$ and the language accepted by $\mathcal{A}_L \cap \overline{\mathcal{B}_\phi}$, respectively.

Consider now the same probabilistic problems but in relation to an observed prefix. Given a sequence s of propositional valuations, we want to analyse only traces t that extend s. Formally, if ϕ is a PLTL$_f$ formula, and s is a finite sequence of propositional valuations, a *most likely trace extending* s is a trace $t = s \cdot u$ such that for every trace $t' = s \cdot u'$, $P_\phi(t') \leq P_\phi(t)$. We want to find all the most likely traces extending s, and their probability. Importantly, a sequence s is not necessarily a most likely trace extending s (that is, extending itself) since the formula ϕ may exclude such a situation from the accepted traces; i.e., it may be impossible to read s and finish in a final state in the automaton recognising ϕ. In other words, the sequence s might not be a full trace. Note, on the other hand, that the set of all finite words over the alphabet of propositional valuations which extend s is recognisable; indeed, a simple concatenation of the universal automaton to the automaton that accepts only s recognises this language. Hence, our previous results answer this question.

Understanding the probabilities of traces extending a given prefix is closely related to the problem of monitoring; that is, controlling that a current execution of a process does not lead to an unwanted scenario. In the classical setting, given a process specification, a (partial) trace may be in one of four states [33]: (i) permanently satisfying, if it currently satisfies the specification and all extensions satisfy it as well; (ii) temporarily satisfying, if it currently satisfies the specification, but some extensions may lead to its violation; (iii) temporarily violating, if it currently violates the specification, but some extensions satisfy it and (iv) permanently violating, if currently and in all extensions it violates the specification. This notion was extended to the probabilistic setting in [1], where for each possible state is assigned a probability—or, more precisely, probabilistic bounds given by the optimistic and pessimistic views.

5 The PLTL$_f^0$ Fragment of PLTL$_f$

We have seen that even the basic task of satisfiability is, in the PLTL$_f$ case, ExpTime-hard on the length of the formula. To mitigate this complexity, we now focus on the fragment of PLTL$_f$ where probabilities can only appear as the top-most temporal constructor of a conjunctive formula. We call this fragment PLTL$_f^0$. Formally, a PLTL$_f^0$ formula is a finite set of expressions of the form $\circledcirc_{\bowtie p}\varphi$, where φ is a classical LTL$_f$ formula, $\bowtie \in \{\leq, \geq, <, >\}$, and $p \in [0, 1]$.[8] We refer to the elements of this set as *probabilistic constraints*. In terms of processes, $\circledcirc_{\bowtie p}\varphi$ expresses that the proportion of traces of the process that satisfy φ is $\bowtie p$. The set of formulas is interpreted as a conjunction of the probabilistic formulas appearing in it; that is, a PLTL$_f^0$ formula is a conjunction of probabilistic constraints. Hence, the formula ϕ_0 from Example 3.3 is a PLTL$_f^0$ formula. Note that in this restricted setting, the probabilistic constructor $\circledcirc$ refers only to the probability of observing a specific LTL$_f$ formula, without a reference to the next point in time. We preserve the same notation, to keep it

[8] Recall that we consider all the standard abbreviations from LTL$_f$. In particular, $\Diamond\varphi \equiv \top U \varphi$, where $\top$ stands for any tautology, and $\Box\varphi \equiv \neg\Diamond\neg\varphi$.

consistent with the general syntax of PLTL$_f$. As we will see later, this restriction allows us to get rid of complex tree-shaped interpretations and focus only on sets of linear interpretations more akin to the classical LTL$_f$ view. This, in particular, helps to improve the efficiency of reasoning methods.

This fragment is interesting for two reasons. On the one hand, we will see that the complexity of reasoning in PLTL$_f^0$ decreases to PSpace, which matches the complexity of the classical (non-probabilistic) LTL$_f$. On the other hand, PLTL$_f^0$ is suited for describing declarative constraints mined from historical log data of business process executions. More precisely, some PLTL$_f^0$ patterns can be readily mined from log data using existing declarative process discovery techniques. For example, if φ describes a property of interest, one can analyse historical process logs to count the proportion of instances where φ is satisfied. In this case, we can assign that proportion as the probability of φ within a PLTL$_f^0$ constraint following the standard frequentist view on probabilities; Section 5.2 provides more details on this idea, summarising from our previous work [1]. Importantly, the flexibility of PLTL$_f^0$ allows us in this case to avoid issues with inconsistency that often arises with classical process mining techniques. Yet, it is important to note that full PLTL$_f$ also has a place in modelling business processes as argued at the end of this section.

5.1 Reasoning in PLTL$_f^0$

When we restrict to the PLTL$_f^0$ fragment, the original superposition semantics of PLTL$_f$ collapse to a version of the multiple-world semantics often used to handle probabilistic logics, in what Halpern calls Type II probabilities [20] (see also [29, 37]). To simplify the presentation of this section, we define a *probabilistic interpretation* as a pair $\mathcal{P} = (\mathcal{I}, P_{\mathcal{I}})$, where $\mathcal{I}$ is a finite set of LTL$_f$ interpretations and $P_{\mathcal{I}}$ is a discrete probability distribution over $\mathcal{I}$. Since $\mathcal{I}$ is finite, we can see $P_{\mathcal{I}}$ as a function $P_{\mathcal{I}} : \mathcal{I} \to [0, 1]$ such that $\sum_{I \in \mathcal{I}} P_{\mathcal{I}}(I) = 1$, and extend it to sets of interpretations by defining for every $\mathcal{J} \subseteq \mathcal{I}$,

$$P_{\mathcal{I}}(\mathcal{J}) := \sum_{I \in \mathcal{J}} P_{\mathcal{I}}(I).$$

Note that the elements of $\mathcal{I}$ are *classical* LTL$_f$ interpretations. Satisfiability of an LTL$_f$ formula by an LTL$_f$ interpretation is defined as usual [13].

Definition 5.1 (PLTL$_f^0$ Model). The probabilistic interpretation $\mathcal{P} = (\mathcal{I}, P_{\mathcal{I}})$ is called a *model* of the PLTL$_f^0$ formula $\Phi = \{\circledcirc_{\bowtie p_i} \varphi_i \mid 1 \leq i \leq n\}$ iff for every i, $1 \leq i \leq n$ it holds that

$$P_{\mathcal{I}}(\{I \in \mathcal{I} \mid I \models \varphi_i\}) \bowtie p_i.$$

In words, $\mathcal{P}$ is a model of Φ if for every probability probabilistic constraint $\circledcirc_{\bowtie p_i} \varphi_i$, the probability of all the models of φ_i is $\bowtie p_i$—that is, the probability distribution is consistent with the constraint. For example, the formula ϕ_0 from Example 3.3 is equivalent to the PLTL$_f^0$ formula $\Phi_0 := \{\circledcirc_{\leq 0.5} \bigcirc a, \circledcirc_{\geq 0.6} a \,\mathcal{U}\, b\}$. Two models of this formula are depicted in Figure 4.

Briefly, the uncertainty of PLTL$_f^0$ formulas appears only at the beginning of the process, after which the execution follows a regular LTL$_f$ execution. Thus, there is no need to branch within the superposition semantics at later times; a model becomes a degenerate tree which only branches at the root. Moreover, as all formulas start with the constructor $\circledcirc$, the root node serves only as an anchor for the PLTL$_f$ semantics. Hence, a model can be represented as a set of classical LTL$_f$ interpretations. Compare the model in Figure 1(c) with Figure 4(a). Interestingly, restricting to PLTL$_f^0$ reduces the complexity of dealing with probabilistic formulas, and allows for simpler algorithms. Consider first the case of deciding whether the PLTL$_f^0$ formula Φ is satisfiable. In practice, this corresponds to verifying whether the class of all possible traces can be divided in

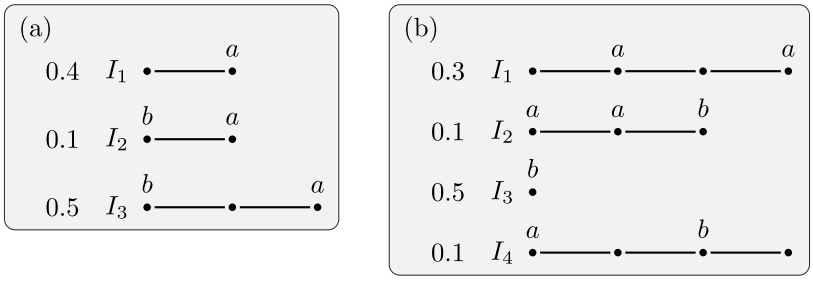

Fig. 4. Two probabilistic models of $\Phi_0 = \{\circledcirc_{\leq 0.5} \bigcirc a, \circledcirc_{\geq 0.6} a \,\mathcal{U}\, b\}$. The probability of each interpretation appears on the left.

such a way that the proportions required by the probabilistic constraints are satisfied. To solve this problem, we may proceed as follows.

Given $\Phi = \{\circledcirc_{\bowtie p_i} \varphi_i \mid 1 \leq i \leq n\}$, we analyse the 2^n possible scenarios of a trace, depending which of the constraints are satisfied and which are violated. More precisely, consider the 2^n sets of constraints in the Cartesian product $\prod_{i=1}^{n} \{\phi_i, \neg\phi_i\}$; i.e., each set chooses for every formula whether it will be satisfied or violated.[9] If each of these sets, seen as the conjunction of the formulas that it contains, is satisfiable, then the input PLTL_f^0 formula is satisfiable as well. On the other hand, if any of these sets is unsatisfiable, it means that it is impossible to build a trace that satisfies that combination of formulas; hence that scenario must be assigned probability 0.

To verify that probabilities for the remaining branches can still be assigned in a manner that is consistent with the probabilistic values appearing in Φ, we build a system of inequalities whose solution space is precisely the valid probability assignments. We consider one variable for each case. For readability, we name these variables $x_0, \ldots, x_{2^n-1}$ using a binary subindex which indicates satisfaction or violation of constraints assuming w.l.o.g. that the constraints are linearly ordered. That is, the subindex is a chain of length n of 0s and 1s; a 0 or a 1 at position i means that $\neg\phi_i$ or ϕ_i is satisfied, respectively. We use the same subindex convention to refer to the sets of constraints S_i; e.g., if Φ contains three formulas, then x_{010} is the variable corresponding to the set $S_{010} = \{\neg\phi_1, \phi_2, \neg\phi_3\}$. The idea is that for each of these variables we will construct a trace which satisfies the formulas defined by its index. Using this information, $\mathcal{L}_\Phi$ is the system of inequalities

$$
\begin{aligned}
x_i &\geq 0 && 0 \leq i < 2^n \\
\sum_{i=0}^{2^n-1} x_i &= 1 && \\
\sum_{j\text{th position is 1}} x_i &\bowtie p_j && 0 \leq j < n \\
x_i &= 0 && \text{if } S_i \text{ is unsatisfiable.}
\end{aligned}
$$

The first two lines guarantee that we assign a non-negative value to each variable, and that their sum is one; in this way, the assignments will produce a probability distribution over the worlds defined by the variables. The third line verifies the probability associated to each constraint in Φ: all the variables that correspond to cases making ϕ_i true should add to be $\bowtie p_i$. The last line ensures that the unsatisfiable cases are never assigned a positive probability. This system of inequalities has a solution iff the PLTL_f^0 formula is satisfiable.

[9]Equivalently, one can consider the 2^n subsets of $\{\varphi_i \mid 1 \leq i \leq n\}$, where an inclusion in the set means that it is satisfied, and the exclusion means that it is violated.

Example 5.2. Let $\Phi_1 := \{\circledcirc_{\leq 0.8}\Diamond a, \circledcirc_{\leq 0.7}\Box(a \rightarrow \Diamond b)\}$. Since Φ_1 has two probabilistic constraints, we build four variables and sets

$$S_{00} := \{\neg\Diamond a, \neg\Box(a \rightarrow \Diamond b)\}, \qquad S_{01} := \{\neg\Diamond a, \Box(a \rightarrow \Diamond b)\},$$
$$S_{10} := \{\Diamond a, \neg\Box(a \rightarrow \Diamond b)\}, \qquad S_{11} := \{\Diamond a, \Box(a \rightarrow \Diamond b)\}.$$

S_{00} is clearly unsatisfiable, but the remaining three sets are satisfiable. The system of inequalities must enforce that x_{00} is 0. Specifically, the system $\mathcal{L}_{\Phi_1}$ is

$$x_{00} = 0 \qquad x_{01} \geq 0 \quad x_{10} \geq 0 \quad x_{11} \geq 0$$
$$x_{00} + x_{01} + x_{10} + x_{11} = 1$$
$$x_{10} + x_{11} \leq 0.8 \qquad x_{01} + x_{11} \leq 0.7.$$

A solution of $\mathcal{L}_{\Phi_1}$ is $x_{00} = 0$, $x_{01} = 0.2$, $x_{10} = 0.3$ and $x_{11} = 0.5$, which yields a probabilistic model of Φ_1 consisting of three interpretations, I_1, I_2, I_3; each interpretation I_i satisfies the constraints in the set S_i and is assigned the probability x_i. An interpretation for S_0 is not needed because these constraints are unsatisfiable (and the combination of formulas is assigned probability 0).

Theorem 5.3. *The PLTL^0_f formula Φ is satisfiable iff $\mathcal{L}_\Phi$ has a solution.*

Proof. Suppose first that $\Phi = \{\circledcirc_{\bowtie p_i}\varphi_i \mid 1 \leq i \leq n\}$ is satisfiable, and let $\mathcal{P} = (\mathcal{I}, P_{\mathcal{I}})$ be a model of Φ. For each $j, 1 \leq j \leq 2^n$, let

$$\mathcal{I}_j := \{I \in \mathcal{I} \mid I \models \bigwedge_{\psi \in S_j} \psi\};$$

that is, $\mathcal{I}_j$ contains all the traces in $\mathcal{I}$ that satisfy the choices of the set S_j. Importantly, the class $\{\mathcal{I}_j \mid 1 \leq j \leq 2^n\}$ forms a partition of $\mathcal{I}$. We now assign the values $x_j := \sum_{I \in \mathcal{I}_j} P_{\mathcal{I}}(I)$. We can see directly that each $x_j \geq 0$ and that for each unsatisfiable choice set S_j, $x_j = 0$. Moreover, as the sets $\mathcal{I}_j$ form a partition of $\mathcal{I}$, and $\mathcal{P}$ is a model of Φ, it follows that $\sum_{j=0}^{2^n} x_j = 1$ and the assignments to the x_js is consistent with all the probabilistic constraints. Hence this assignment is a solution of $\mathcal{L}_\Phi$.

Conversely, let $\{x_j \mid 1 \leq j \leq 2^n\}$ be a solution of $\mathcal{L}_\Phi$. For each $x_j > 0$ consider an LTL$_f$ model I_j of S_j, which exists because by construction x_j can only be positive if S_j is consistent. Then the pair $(\mathcal{I}, \mathcal{P}_{\mathcal{I}})$ where $\mathcal{I} = \{I_j \mid x_j > 0\}$ and $\mathcal{P}_{\mathcal{I}}(I_j) = x_j$ for each $x_j > 0$ is a PLTL^0_f model of Φ. □

To construct the system of inequalities $\mathcal{L}_\Phi$, one must solve 2^n LTL$_f$ satisfiability tests, each one requiring polynomial space [44]. Solving this system of inequalities requires polynomial time on the number of variables; i.e., exponential on n. Overall, it needs exponential time on n, but only polynomial space on the length of Φ. Thus we get the following result.

Theorem 5.4. *PLTL^0_f satisfiability is decidable in exponential time on the number of probabilistic formulas, but in polynomial space on their total length.*

In particular, if the number of formulas in Φ is bounded, satisfiability is PSpace-complete, improving the ExpTime lower bound for general PLTL$_f$ (see Theorem 3.10). We are more interested in deducing (probabilistic) guarantees of a process that follows the constraints in a formula; and more importantly of traces being observed over them. Recall that in our semantics any execution is possible as long as there is no evidence to the contrary. In PLTL^0_f the uncertainty is stated at the beginning; that is, we do not know which of the formulas $\phi_1, \ldots, \phi_n$ are satisfied, but once a trace has chosen its path, it remains in it without further uncertainty arising later on. Continuing with the metaphor from physics, the superposed possible states collapse to one state once the execution of the trace is observed.

As before, we follow an optimistic approach and try to find the most likely scenarios and traces that fit the constraints in the formula. Recall that the solution space of $\mathcal{L}_\Phi$ yields the probability assignments that can be consistently given to the LTL_f interpretations appearing in a probabilistic model according to the constraints that it satisfies. Thus, maximising the value of a variable x_i yields the maximum probability that the set S_i may be assigned in a model.

For each $i, 0 \leq i < 2^n$, let $\max_i$ be the solution of maximising x_i subject to $\mathcal{L}_\Phi$. Note—as before—that each variable is maximised independently of the others, and hence the values $\max_i$ do not form a probability distribution *per se*; their sum may be greater (but never lower) than 1. The values $\max_i$ express the *optimistic* position of assigning the highest possible probability to the traces satisfying S_i. For the formula Φ_1 from Example 5.2, the answers to the maximisation problems for the different variables correspond precisely to the values in the solution presented; namely, $\max_{00} = 0$, $\max_{01} = 0.2$, $\max_{10} = 0.3$ and $\max_{11} = 0.5$.

The question of finding the mlts or simply the *most likely scenario* can be answered using the values $\max_i$. Take the indices j where $\max_j$ is the largest among all variables; i.e., $j \in \operatorname{argmax}\{\max_i \mid 0 \leq i < 2^n\}$. Each S_j is a most likely scenario, and every trace satisfying S_j is an mlt, with probability $\max_j$. In our example, S_{11} is the most likely scenario; i.e, we expect to observe a trace satisfying $\Diamond a$ and $\Box(a \rightarrow \Diamond b)$.

If Φ represents a process, the mlts are those that we would expect to observe in an execution of the process in the absence of other information. In general, it is more interesting to predict the future behaviour of the process, given some observation of its first steps. Given a (partial) trace t, corresponding to the prefix of a full process, we want to find the most likely scenario where t can happen, and predict the potential future evolution of the trace. Given a set S of LTL_f formulas and a prefix t, S *accepts* t (denoted by $S \Vdash t$) iff there is a suffix s such that $S \models t \cdot s$. In words, S accepts a prefix if it can be extended into a trace that satisfies all the conditions in S. To find the most likely scenario accepting a prefix, and a suffix extending it to a successful trace, we generalise the idea described for the case without prefix.

Let $\mathsf{acc}(t) := \{i \mid S_i \Vdash t\}$ be the set of indices j such that S_j accepts t, and let $j \in \operatorname{argmax}_{i \in \mathsf{acc}(t)} \{\max_i\}$ be an index with the maximum value in the set of solutions from the maximisation problems of $\mathcal{L}_\Phi$. Then, S_j is a most likely scenario given t, and any trace extending t accepted by S_j is an mlt. For Φ_1 in Example 5.2, the most likely scenario for any finite prefix t is always S_{11}, and $t \cdot ab$ is always a trace accepted by this set. In general, however, the most likely scenario may change as a prefix grows.

Example 5.5. Let $\Psi_1 := \{\circledcirc_{\leq 0.5} \Diamond a, \circledcirc_{\leq 0.6} \Box(a \rightarrow \Diamond b)\}$, which is very similar to the PLTL_f^0 formula Φ_1 from Example 5.2, but with different probabilities. We get $\max_{00} = 0$, $\max_{01} = 0.5$, $\max_{10} = 0.4$ and $\max_{11} = 0.1$. For the empty prefix ε and the prefix $\neg a$, the most likely scenario is S_{01}, which holds with probability 0.5, and a most likely continuation would append them with a finite number of observations of $\neg a$. If at the second point in time we observe a (making the prefix $\neg a \cdot a$) then S_{01} does not accept this prefix anymore, and the most likely scenario becomes S_{10}: we will eventually observe an a after which b will never be observed anymore.

The method for finding the most likely scenario is formalised in Algorithm 1, where $\max_{-1} := 0$. Note that an expensive part of this algorithm is the computation of the values $\max_i$. However, this computation can be made offline, as a preprocessing step before the algorithm is called, as these values remain invariant for any call. A second point to consider is that the set J monotonically decreases as the trace t grows. More precisely, for every t, s, if $S \Vdash t \cdot s$, then $S \Vdash t$. Hence, if we are monitoring the evolution of a process, trying to find out the most likely continuation of the currently observed trace, then after every newly observed step, we only need to update the set J to remove those S_is not accepting the prefix anymore. Finally, to avoid unnecessary tests, we can exclude from the *for* loop all indices i where $\max_i = 0$: $\max_i = 0$ means that the system should not

Algorithm 1: Most Likely Scenario for t Over Φ

Data: $\Phi=\{\circledcirc_{\bowtie p_i}\varphi_i | 1\leq i\leq n\}$ PLTL_f^0 formula, t prefix
Result: Index of the most likely scenario for t in Φ
$mls \leftarrow -1$
for $0 \leq i < 2^n$ **do**
  compute $\max_i$
  if $S_i \Vdash t$ *and* $\max_i > \max_{mls}$ **then**
    $mls \leftarrow i$
  end
end
Return mls

observe any trace satisfying S_i. If the most likely scenario is one that has probability 0, then the observed prefix is violating the conditions described by Φ.

Proposition 5.6. *Algorithm 1 returns the index of the most likely scenario, given a prefix.*

Interestingly, assuming that all the values $\max_i$ have been computed before, Algorithm 1 can be executed to use only polynomial space. The information to control the *for* loop requires at most n bits, and the two tests within this loop require polynomial space.

Note that finding the probability of the most likely scenario (and trace) is akin to monitoring agreement with a model. Analogously, one can extend the task to monitoring a complex PLTL_f property ψ. For the maximum likelihood of accepting ψ, we use Algorithm 1, but now considering whether $S_i \cup \{\psi\} \Vdash t$; i.e., finding the scenarios where ψ may still be satisfied given the knowledge of t. This is analogous to the notion of *eventual satisfaction* in monitoring. Other notions like *current satisfaction*, or *permanent satisfaction* can be dealt with accordingly, modifying the notion of acceptability of a trace w.r.t. a set of LTL_f constraints [9].

5.2 Discovering PLTL_f^0 Patterns from Event Log Data

PLTL_f^0 formulas can be automatically mined from event log data using state-of-the-art *declarative process discovery* algorithms within *process mining*. Process mining focuses on the continuous improvement of business processes based on factual data [45]. Such data are stored in a so-called *event log*, where each event refers to an *activity* (a well-defined step in a process) and is related to a *case* (a *process instance*). Events in a case are *ordered* and seen as an execution (or *trace*) of the process. A core process mining task is *process discovery*, which learns a process model that reproduces the traces contained in the log. In declarative process discovery, the target model is specified using rules (or constraints), like the LTL_f patterns adopted by the Declare process modelling language [39].

Maggi et al. [30] developed a two-phase method to automatically infer Declare constraints from event logs. In the first phase, candidate constraints to be mined are generated by an algorithm called Apriori. This algorithm returns frequent activity sets indicating a high correlation between activities involved in an activity set. Highly correlated sets are used to instantiate, in any possible ways, the Declare patterns. For example, considering the frequent activity set $\{a, b\}$ and the Declare pattern of `response`, the two LTL_f constraints $\Box(a \rightarrow \Diamond b)$ and $\Box(b \rightarrow \Diamond a)$ are generated. In the second phase, the set of so-generated constraints is filtered by retaining only "relevant" constraints, where relevance is measured using metrics such as that of *support*: the proportion of traces satisfying the constraint.

What makes Apriori interesing in our setting is that support can be interpreted as the constraint probability: the discovery of LTL_f formula φ with support p (that is, appearing in $100p\%$ of the traces) can be interpreted as the discovery of the PLTL_f^0 formula $\circledcirc_{=p}\varphi$.

In a nutshell, we are using a standard frequentist approach to measure the probability of observing a specific behaviour in a process: the probability is exactly the proportion of executions that showcase that behaviour within all our available observations. This allows us to assign probabilities to each property of interest in the description of a process model. The approach was fully developed and studied in [1].

5.3 PLTL_f^0 and Declare

PLTL_f^0 is of particular interest in the area of **business process management (BPM)** [18] and process mining [45]. BPM is a discipline at the intersection between operations management and computer science, whose goal is to manage the operational processes executed inside an organisation throughout their whole lifecycle. Process mining enhances BPM with a set of data- and model-driven techniques to discover processes from actual execution data, check the conformance of actual executions with reference process models and enhance process models with features and indicators extracted from data.

Within the scope of BPM and process mining, an important choice concerns which formalism is selected to represent processes. Petri net-based formalisms were traditionally used, due to their ability of representing key process modelling patterns such as sequencing, choice and concurrency. A widely investigated alternative, particularly useful when capturing flexible processes [40], consists in adopting declarative process modelling languages, where the execution traces supported by the process are implicitly characterised as all finite traces that satisfy a set of temporal constraints. One of the most studied language in this direction is Declare, which uses LTL_f as underlying constraint specification formalism [14].

Declare comes with a set of templates providing LTL_f patterns for specifying constraints. A Declare specification consists of a set of constraints, obtained by grounding such templates on concrete activities. A trace then belongs to the specification if it satisfies *all* the constraints in the set. This crisp semantics is often too restrictive, as it makes it impossible to capture best practices (i.e., constraints that usually hold, but are sometimes violated), outliers (i.e., constraints that capture few, relevant executions), or constraints involving uncontrollable parties (such as references to external customers and related uncertainty). In this respect, PLTL_f^0 is a natural candidate for infusing these types of constraints in Declare. In fact, a probabilistic extension of Declare, called probDeclare, was proposed in [32], using PLTL_f^0 as underlying logic. The reasoning services defined in Section 5.1 to deal with scenarios and their likelihood, paired with standard automata-theoretic techniques to tackle each distinct scenario separately, were used in the context of probDeclare, ranging from model analysis to probabilistic, declarative process mining [1]. The main task considered are *specification consistency*, which corresponds to satisfiability in PLTL_f^0; *discovery*; *conformance checking* and *monitoring*. Next we briefly explain the last three tasks.

Discovery refers to the task where, starting from an event log containing (a multiset of) traces representing recorded executions of the process, the goal is to learn a probDeclare specification containing relevant constraints, with associated probabilities. Interestingly, probabilities are a by-product of support metrics computed by standard algorithms for Declare discovery, making it possible to directly lift them to learn a class of PLTL_f^0 formulae whose temporal component is one of the Declare patterns.

Conformance Checking has the goal of understanding whether a given observed trace belongs to a reference specification. In probDeclare, this task can be refined by checking to which scenario the trace belongs. If the probability associated to that scenario is 0, then the trace is deemed as non-conforming; otherwise, the trace is deemed as conforming, and the likelihood of that scenario can be used to determine whether the trace is a common one, or an outlier.

Monitoring is the task whose goal is to dynamically check conformance for an evolving trace under observation. In particular, for LTL_f and Declare, anticipatory forms of monitoring have been studied, by recasting runtime verification techniques in the context of finite traces. The underlying idea is to judge the monitoring status of a formula considering not only the trace prefix observed so far, but also the infinitely many, possible finite-length continuations. In the case of probDeclare, while a complete trace belongs to one and only one scenario, a trace prefix may in principle map to multiple candidate scenarios. This calls for aggregating the monitoring statuses of each such scenarios, as well as their respective likelihoods.

5.4 PLTL_f in BPM

We conclude by showing that the full language PLTL_f, beyond PLTL_f^0, can also be of interest in modelling complex process properties within the realm of BPM. Recall that in PLTL_f^0, probabilities can appear only as the outer-most constructor. Hence, it is possible to express conditions like "*within an execution, there is a probability at least p that if a product is shipped, it will be delivered*" through the PLTL_f^0 formula $\circledcirc_{\geq p}\Box(\mathsf{ship} \rightarrow \Diamond\ \mathsf{deliver})$.

Suppose that the business process foresees several shipments within one single execution (trace); e.g., if an order is divided into several packages. According to our semantics, the PLTL_f^0 formula above expresses that *all* the shipments will be delivered with probability at least p. But in practice, one would actually like to express that *each* shipment will be delivered with that probability, as formalised by the PLTL_f formula $\Box(\mathsf{ship} \rightarrow \circledcirc_{\geq p}\Diamond\ \mathsf{deliver})$, which is not expressible in PLTL_f^0.

6 Conclusions

In this work, we introduced a new PLTL, called PLTL_f, which is based on a novel superposition semantics, along with a sublogic (PLTL_f^0) where this semantics collapses to the standard multiple-world approach commonly used in probabilistic logics. PLTL_f and PLTL_f^0 are specifically crafted for predicating about uncertainty in dynamic systems whose executions eventually finish; hence the emphasis on finite time. We studied the main properties of the logics, and provided automata-theoretic methods—based on tree automata and on weighted automata—for extracting relevant information from them.

The study of these probabilistic extensions of LTL_f opens new possibilities for representing and reasoning about processes in real-life scenarios, where uncertainty is unavoidable, but can still be measured through statistical techniques. This yields a more robust and realistic analysis of business processes and their executions than what can be obtained through purely logical approaches. In particular, PLTL_f (and by extension also its restricted version PLTL_f^0) are adequate languages for modelling uncertainty in process executions.

The algorithms described for PLTL_f^0 have already been implemented and applied to the declarative modelling and analysis of business processes, considering in particular monitoring and conformance checking [1]. As future work we plan to further understand the formal properties and the computational complexity of reasoning with this language and its sub-logics, of which other variants will be presented. In addition, we aim at providing explanation capabilities to LTL_f and PLTL_f.

References

[1] Anti Alman, Fabrizio Maria Maggi, Marco Montali, and Rafael Peñaloza. 2022. Probabilistic declarative process mining. *Inf. Syst.* 109 (2022), 102033. DOI: https://doi.org/10.1016/J.IS.2022.102033

[2] Franz Baader, Jan Hladik, and Rafael Peñaloza. 2008. Automata can show PSpace results for description logics. *Inf. Comput.* 206, 9–10 (2008), 1045–1056.

[3] Franz Baader and Rafael Peñaloza. 2010. Automata-based axiom pinpointing. *J. Autom. Reason.* 45, 2 (2010), 91–129. DOI: https://doi.org/10.1007/S10817-010-9181-2

[4] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press.

[5] Ronen I. Brafman, Giuseppe De Giacomo, and Fabio Patrizi. 2018. LTLf/LDLf Non-Markovian rewards. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI '18)*. AAAI Press, 1771–1778.

[6] Josep Carmona, Boudewijn F. van Dongen, Andreas Solti, and Matthias Weidlich. 2018. *Conformance Checking—Relating Processes and Models*. Springer.

[7] Miroslav Chodil and Antonín Kučera. 2024. The finite satisfiability problem for PCTL is undecidable. In *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '24)*. ACM. DOI: https://doi.org/10.1145/3661814.3662145

[8] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. 2007. Tree automata techniques and applications. Retrieved October 12, 2007 from http://www.grappa.univ-lille3.fr/tata

[9] G. De Giacomo, R. De Masellis, M. Grasso, F. M. Maggi, and M. Montali. 2014. Monitoring business metaconstraints based on LTL & LDL for finite traces. In *Proceedings of the 12th International Conference on Business Process Management (*BPM '14). LNCS, Vol. 8659, Springer, 1–17.

[10] Giuseppe De Giacomo, Riccardo De Masellis, and Marco Montali. 2014. Reasoning on LTL on finite traces: Insensitivity to infiniteness. In *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI Press, 1027–1033.

[11] Giuseppe De Giacomo, Fabrizio Maria Maggi, Andrea Marrella, and Fabio Patrizi. 2017. On the disruptive effectiveness of automated planning for LTL$_f$-based trace alignment. In *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI Press, 3555–3561.

[12] Giuseppe De Giacomo and Sasha Rubin. 2018. Automata-theoretic foundations of FOND planning for LTLf and LDLf goals. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI '18)*, 4729–4735.

[13] Giuseppe De Giacomo and Moshe Y. Vardi. 2013. Linear temporal logic and linear dynamic logic on finite traces. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI '13)*. AAAI Press, 854–860.

[14] Claudio Di Ciccio and Marco Montali. 2022. Declarative process specifications: Reasoning, discovery, monitoring. In *Process Mining Handbook*. Wil M. P. van der Aalst and Josep Carmona (Eds.), Lecture Notes in Business Information Processing, Vol. 448. Springer, 108–152. DOI: https://doi.org/10.1007/978-3-031-08848-3_4

[15] Dragan Doder and Aleksandar Perović. 2020. Probabilistic temporal logics. In *Probabilistic Extensions of Various Logical Systems*. Zoran Ognjanović (Ed.), Springer International Publishing, Cham, 71–108. DOI: https://doi.org/10.1007/978-3-030-52954-3_3

[16] Rodney G. Downey and M. R. Fellows. 2012. *Parameterized Complexity*. Springer.

[17] Manfred Droste, Werner Kuich, and Heiko Vogler. 2009. *Handbook of Weighted Automata* (1st ed.). Springer.

[18] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. 2018. *Fundamentals of Business Process Management* (2nd. ed.). Springer. DOI: https://doi.org/10.1007/978-3-662-56509-4

[19] Alfonso Gerevini, Patrik Haslum, Derek Long, Alessandro Saetti, and Yannis Dimopoulos. 2009. Deterministic planning in the fifth International planning competition: PDDL3 and experimental evaluation of the planners. *Artif. Intell.* 173, 5–6 (2009), 619–668.

[20] Joseph Y. Halpern. 1990. An analysis of first-order logics of probability. *Artif. Intell.* 46, 3 (1990), 311–350. DOI: https://doi.org/10.1016/0004-3702(90)90019-V

[21] Hans Hansson and Bengt Jonsson. 1994. A logic for reasoning about time and reliability. *Form. Asp. Comput.* 6, 5 (Sep. 1994), 512–535. 0934–5043 DOI: https://doi.org/10.1007/BF01211866

[22] Jan Hladik and Rafael Peñaloza. 2006. PSPACE automata for description logics. In *Proceedings of the 2006 International Workshop on Description Logics (DL '06)*. Bijan Parsia, Ulrike Sattler, and David Toman (Eds.), CEUR-WS, Vol. 189, The Lake District, UK.

[23] Jan Hladik and Ulrike Sattler. 2003. A translation of looping alternating automata Into description logics. In *Proceedings of the the 19th Conference on Automated Deduction (CADE '19)*. Franz Baader (Ed.). Springer, Berlin, 90–105.

[24] John E. Hopcroft and Jeff D. Ullman. 1979. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company.

[25] N. Karmarkar. 1984. A new polynomial-time algorithm for linear programming *(STOC '84)*. ACM, New York, NY, 302–311. DOI: https://doi.org/10.1145/800057.808695

[26] L. G. Khachiyan. 1980. Polynomial algorithms in linear programming. *U. S. S. R. Comput. Math. and Math. Phys.* 20, 1 (1980), 53–72. DOI: https://doi.org/10.1016/0041-5553(80)90061-0
[27] Savas Konur. 2010. Real-time and probabilistic temporal logics: An overview. arXiv:1005.3200. Retrieved from http://arxiv.org/abs/1005.3200
[28] Alisa Kovtunova and Rafael Peñaloza. 2018. Cutting diamonds: A temporal logic with probabilistic distributions. In *Proceedings of the 16th International Conference on Principles of Knowledge Representation and Reasoning (KR '18)*. AAAI Press, 561–570.
[29] Thomas Lukasiewicz. 1998. Probabilistic logic programming. In *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI '98)*. Henri Prade (Ed.), John Wiley and Sons, 388–392.
[30] Fabrizio Maria Maggi, R. P. Jagadeesh Chandra Bose, and Wil M. P. van der Aalst. 2012. Efficient discovery of understandable declarative process models from event logs. In *Proceedings of the 24th International Conference on Advanced Information Systems Engineering (CAiSE '12)*. LNCS, Vol. 7328,. Springer, 270–285.
[31] Fabrizio M. Maggi, Marco Montali, and Rafael Peñaloza. 2020. Temporal logics over finite traces with uncertainty. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI '20)*. AAAI Press, 10218–10225. Retrieved from https://ojs.aaai.org/index.php/AAAI/article/view/6583
[32] Fabrizio Maria Maggi, Marco Montali, Rafael Peñaloza, and Anti Alman. 2020. Extending temporal business constraints with uncertainty. In *Proceedings of the 18th International Conference on Business Process Management (BPM '20)*. Dirk Fahland, Chiara Ghidini, Jörg Becker, and Marlon Dumas (Eds.), Lecture Notes in Computer Science, Vol. 12168, Springer, 35–54. DOI: https://doi.org/10.1007/978-3-030-58666-9_3
[33] Fabrizio Maria Maggi, Marco Montali, Michael Westergaard, and Wil M. P. van der Aalst. 2011. Monitoring business constraints with linear temporal logic: An approach based on colored automata. In *Proceedings of the Business Process Management (BPM '11)*. LNCS, Vol. 6896, Springer, 132–147.
[34] S. Meyn and R. L. Tweedie. 2009. *Markov Chains and Stochastic Stability*. Cambridge University Press. Retrieved from https://books.google.it/books?id=Md7RnYEPkJwC
[35] Marco Montali. 2010. *Specification and Verification of Declarative Open Interaction Models: a Logic-Based Approach*. LNBIP, Vol. 56. Springer.
[36] Luís Morão. 2011. *Probabilistic Distributed Temporal Logic*. Master Thesis. Universidade Técnica de Lisboa, Portugal.
[37] Nils J. Nilsson. 1986. Probabilistic logic. *Artificial Intelligence* 28, 1 (1986), 71–87. DOI: https://doi.org/10.1016/0004-3702(86)90031-7
[38] Zoran Ognjanovic. 2006. Discrete linear-time probabilistic logics: Completeness, decidability and complexity. *J. Log. Comput.* 16, 2 (2006), 257–285. DOI: https://doi.org/10.1093/logcom/exi077
[39] Maja Pesic, Helen Schonenberg, and Wil M. P. van der Aalst. 2007. DECLARE: Full support for loosely-structured processes. In *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC '07)*. IEEE Computer Society, 287–300.
[40] Manfred Reichert and Barbara Weber. 2012. *Enabling Flexibility in Process-Aware Information Systems—Challenges, Methods, Technologies*. Springer. DOI: https://doi.org/10.1007/978-3-642-30409-5
[41] Walter J. Savitch. 1970. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. System Sci.* 4, 2 (1970), 177–192. DOI: https://doi.org/10.1016/S0022-0000(70)80006-X
[42] Daniel J. Segalman, Matthew R. Brake, Lawrence A. Bergman, Alexander F. Vakakis, and Kai Willner. 2013. Epistemic and aleatoric uncertainty in modeling. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Vol. 55997. American Society of Mechanical Engineers.
[43] Helmut Seidl. 1994. Haskell overloading is DEXPTIME-complete. *Inform. Process. Lett.* 52, 2 (1994), 57–60. DOI: https://doi.org/10.1016/0020-0190(94)00130-8
[44] A. P. Sistla and E. M. Clarke. 1985. The complexity of propositional linear temporal logics. *J. ACM* 32, 3 (1985), 733–749. DOI: https://doi.org/10.1145/3828.3837
[45] Wil M. P. van der Aalst. 2016. *Process Mining - Data Science in Action* (2nd ed.). Springer.
[46] M. Y. Vardi and P. Wolper. 1994. Reasoning about infinite computations. *Information and Computation* 115, 1 (1994), 1–37. DOI: https://doi.org/10.1006/inco.1994.1092
[47] Moshe Y. Vardi and Pierre Wolper. 1986. Automata-theoretic techniques for modal logics of programs. *J. Comput. Syst. Sci.* 32, 2 (1986), 183–221. DOI: https://doi.org/10.1016/0022-0000(86)90026-7
[48] Bruno Woltzenlogel Paleo. 2016. An expressive probabilistic temporal logic. arXiv:1603.07453. Retrieved from http://arxiv.org/abs/1603.07453

Received 8 February 2024; revised 23 September 2024; accepted 9 January 2025