HyperLDLf: a Logic for Checking Properties of Finite Traces Process Logs

Giuseppe De Giacomo,[1]**, Marco Montali,** [2]**, Paolo Felli** [2]**, Giuseppe Perelli** [1]

[1] Sapienza University of Rome

[2] Free University of Bozen-Bolzano

degiacomo@diag.uniroma1.it, {montali,pfelli}@inf.unibz.it, perelli@diag.uniroma1.it

Abstract

Temporal logics over finite traces, such as LTL_f and its extension LDL_f, have been adopted in several areas, including Business Process Management (BPM), to check properties of processes whose executions have an unbounded, but finite, length. These logics express properties of single traces in isolation, however, especially in BPM it is also of interest to express properties over the entire log, i.e., properties that relate multiple traces of the log at once. In the case of infinite-traces, HyperLTL has been proposed to express these "hyper" properties. In this paper, motivated by BPM, we introduce HyperLDL_f, a logic that extends LDL_f with the hyper features of HyperLTL. We provide a sound, complete and computationally optimal technique, based on DFAs manipulation, for the model checking problem in the relevant case where the set of traces (i.e., the log) is a regular language. We illustrate how this form of model checking can be used to specify and verify sophisticated properties within BPM and process mining.

1 Introduction

Temporal logics traditionally assume that the traces generated by the dynamic system of interest have infinite length. This does not match a variety of different settings where traces have an unbounded, but finite, length, such as planning [Baier and McIlraith, 2006; Baier *et al.*, 2007; Gerevini *et al.*, 2009] and Business Process Management (BPM) [Dumas *et al.*, 2018; van der Aalst and others, 2011].

To specify and reason about temporal properties of finite traces, De Giacomo and Vardi (2013) studied two finite-trace variants of Linear Temporal Logic and Linear Dynamic Logic, namely LTL_f and LDL_f. These logics turned out to be particularly important in BPM: LTL_f has become the formal basis for one of the most studied declarative process modeling notations [Pesic *et al.*, 2007], while LDL_f has been employed to realize advanced forms of process monitoring and runtime verification [De Giacomo *et al.*, 2014]. Notably, these logics are not only interesting from the semantical point of view, but also bring an operational advantage over their infinite-trace counterparts, since reasoning can be carried out by directly manipulating finite-state automata, without appealing to automata over infinite objects [De Giacomo and Vardi, 2013; De Giacomo *et al.*, 2014].

LTL_f and LDL_f express properties of *single traces in isolation*. However, especially in BPM it is also of interest to express properties that relate multiple traces at once (see, e.g., [van der Aalst *et al.*, 2004; Weidlich *et al.*, 2011; Rinderle-Ma *et al.*, 2016]). Two specific settings arise in this spectrum. In the first setting, such traces belong to the overall, possibly *infinite log* of traces representing all the valid executions of a given process model. In this case, traces are implicitly represented in the form of a finite-state transition system or a bounded Petri net interpreted under interleaving semantics, which guarantee that *the induced log forms a regular language*. These logs not only capture any finite concrete process log, but also capture logs that can be inferred through sampling by model-learning and process mining techniques [Vaandrager, 2017; van der Aalst, 2011]. In the second setting, which is the typical one in *business process management* [van der Aalst and others, 2011], traces belong to a *finite event log* explicitly storing a set of observed process executions, representing a factual sample of all the traces that belong to an unknown process, which again is assumed to be regular.

In the case of infinite-traces, HyperLTL has been proposed to express "hyper" properties that quantify over multiple traces at once [Clarkson *et al.*, 2014]. In this paper, motivated by BPM we introduce HyperLDL_f and its fragment HyperLTL_f: two extensions of LDL_f and LTL_f, respectively, that have the ability to express and verify *hyper-properties of finite traces*.

To handle the analysis of (finite and infinite) logs, we in particular focus on the very significant task of *model checking*: the task of verifying HyperLDL_f formulas over a regular log, i.e., a (possibly infinite) set of traces forming a regular language. We study the HyperLDL_f model checking problem in the key setting where the regular log is compactly described by a deterministic finite automaton. Specifically, we show that model-checking a HyperLDL_f formula with k alternations of trace quantifiers is k-EXPSPACE-complete. Notably, we show the upper bound by presenting a sound and complete algorithm based on manipulation of regular automata that retains the good, practical computational characteristics already exploited in several reasoning tasks for LTL_f/LDL_f. Finally, we go back to our original motivation

and discuss how HyperLTL$_f$ and HyperLDL$_f$ can be used to specify and verify relevant, sophisticated properties within BPM and process mining.

2 Automata and Languages

We recall some basic notions on alternating, nondeterministic, universal, and deterministic finite automata. All these automata capture regular languages but they have different characteristics that we will exploited later [Davis *et al.*, 1994]. For a given set X, $\mathcal{B}^+(X)$ is the set of *positive Boolean* formulas over X, that is Boolean formulas built from elements of X using $\wedge$ and $\vee$, where we also allow true and false. A subset $Y \subseteq X$ satisfies a Boolean formula $\chi \in \mathcal{B}^+(X)$ if the truth assignment assigning true to the elements in Y and false to the elements in $X \setminus Y$ satisfies χ. Given an alphabet Σ, a *finite word* over it is a finite sequence $w = \sigma_0 \cdot \sigma_1 \cdot \ldots \cdot \sigma_n$ of letters in Σ. By $|w| = n+1$ we denote the length of w. Moreover, by Σ^* we denote the set of all possible finite words over Σ. An *alternating finite automaton* (AFA) on finite words is a tuple $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F \rangle$ where Σ is the (finite) input alphabet, Q is a finite set of states with $q_0 \in Q$ being initial state, $\delta : Q \times \Sigma \to \mathcal{B}^+(Q)$ is a transition function and $F \subseteq Q$ is a set of final states. Intuitively, the transition function $\delta(q, \sigma)$ describes the possible configurations that $\mathcal{A}$ can reach when reading the symbol σ while being in state q. The automaton $\mathcal{A}$ is *nondeterministic* (NFA), denoted $\mathcal{N}$, or *universal* (UFA), denoted $\mathcal{U}$, if $\delta(q, \sigma)$ contains only disjunctions or conjunctions, respectively. Finally, it is *deterministic* (DFA), denoted $\mathcal{D}$, if the transition function maps each pair (q, σ) to a single state q'.

The concepts of *run*, *accepting run*, and *accepted word* for an automaton $\mathcal{A}$ are quite standard and we refer to [Leiss, 1985; Davis *et al.*, 1994] for their formal definition. Let $\mathcal{L}(\mathcal{A})$ be the set of words accepted by the automaton $\mathcal{A}$. Recall that every automaton $\mathcal{A}$ can be turned into an equivalent nondeterministic one $\mathcal{N}$, that is, such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{N})$. Such operation is called *nondeterminization* and by $\mathsf{Ndet}(\mathcal{A})$ we denote the nondeterministic equivalent automaton. Analogously, we can turn an automaton $\mathcal{A}$ into a universal one, denoted $\mathsf{Univ}(\mathcal{A})$. Such operation is called *universalization*. Moreover, recall that if $\mathcal{A}$ is either alternating or universal, $\mathsf{Ndet}(\mathcal{A})$ can be of size exponential w.r.t. $|\mathcal{A}|$. Analogously, if $\mathcal{A}$ is either alternating or nondeterministic, $\mathsf{Univ}(\mathcal{A})$ can be of size exponential w.r.t. $|\mathcal{A}|$ [Chandra *et al.*, 1981; Davis *et al.*, 1994].

3 HyperLDL$_f$ over Finite Traces

We start by defining the hyper-version of LDL$_f$. We refer to [De Giacomo and Vardi, 2013] for details on LDL$_f$ itself.

Definition 1 (HyperLDL$_f$ syntax)**.** *For a set* AP *containing atomic propositions* p *and a set* V *of trace variables* π*, the syntax of HyperLDL$_f$ is the following:*

$$\varphi := \psi \mid \exists \pi \varphi \mid \forall \pi \varphi$$
$$\psi := \mathsf{tt} \mid \mathsf{ff} \mid \neg\psi \mid \psi \wedge \psi \mid \psi \vee \psi \mid \langle \rho \rangle \psi \mid [\rho]\psi$$
$$\rho := \phi \mid \psi? \mid \rho + \rho \mid \rho; \rho \mid \rho^*$$
$$\phi := p_\pi \mid \neg p_\pi \mid \phi \wedge \phi \mid \phi \vee \phi$$

Observe that the atomic propositions occurring in a HyperLDL$_f$ formula are typed over the trace (variable) on which they are evaluated. This is also the case for the propositional true and false predicate. In particular, we have $\mathsf{true}_\pi \doteq p_\pi \vee \neg p_\pi$ and $\mathsf{false}_\pi \doteq \neg \mathsf{true}_\pi$ for every trace variable occurring in a HyperLDL$_f$ formula. Analogously, by $\mathsf{true}_P \doteq \bigwedge_{\pi \in P} \mathsf{true}_\pi$ and $\mathsf{false}_P \doteq \neg \mathsf{true}_P$ we denote the propositional true and false over a set of traces. For convenience we use the abbreviation $\mathsf{finished}_\pi \doteq [\mathsf{true}_\pi]\mathsf{ff}$ to specify that a trace π is finished, and $\mathsf{last}_\pi \doteq \langle \mathsf{true}_\pi \rangle \mathsf{finished}_\pi$ to specify the last element of the trace.

A *log* $\mathsf{Log} \subseteq (2^{\mathrm{AP}})^*$ is a (possibly infinite) subset of traces over the powerset of AP. Note that an element P of 2^{AP} can be regarded also as a truth-assignment of AP, where every element in P takes the value true and every element not in P takes the value false. For a trace $t \in \mathsf{Log}$, $|t|$ denotes the length of t. By $t[i]$ we denote the i-th element over t, with $t[0]$ and $t[|t|-1]$ being the first and last elements, respectively. For every $i, j \in \mathbb{N}$ with $i \leq j$, by $t[i, j]$, we denote the subtrace obtained from t by taking the elements form $t[i]$ to $t[j-1]$. For the case of $|t| \leq i$, the trace $t[i, j] = \epsilon$ is the *empty* trace. For the case $i < |t| \leq j$, the trace $t[i, j] = t[i, |t|-1]$ is the suffix of t starting from the i-th element. We also denote the suffix of t from element i with $t[i, \ldots]$.

A *trace assignment* (or simply assignment) $\Pi : V \to \mathsf{Log}$ is a function mapping trace variables V to concrete traces in Log. By $\Pi[\pi \leftarrow t]$ we denote the trace assignment that adheres with Π, except that $\Pi[\pi \leftarrow t](\pi) = t$, that is, t is assigned to π. By $\Pi[i, j]$, for every $i, j \in \mathbb{N}$, we denote the projection of assignment Π over the subtraces between i and j, i.e., such that $\Pi[i, j](\pi) = \Pi(\pi)[i, j]$ for each $\pi \in V$. By $\Pi[i, \ldots]$, for every $i \in \mathbb{N}$, we denote the projection of Π over the suffix from i, i.e., such that $\Pi[i, \ldots](\pi) = \Pi(\pi)[i, \ldots]$ for each $\pi \in V$.

The semantics of a HyperLDL$_f$ formula over a set Log of traces, a trace assignment Π, and a natural number $i \in \mathbb{N}$ is given as follows:

- $\mathsf{Log}, \Pi, i \models \exists \pi \varphi$ if there exists $t \in \mathsf{Log}$ such that $\mathsf{Log}, \Pi[\pi \leftarrow t], i \models \varphi$;
- $\mathsf{Log}, \Pi, i \models \forall \pi \varphi$ if for each $t \in \mathsf{Log}$ it holds that $\mathsf{Log}, \Pi[\pi \leftarrow t], i \models \varphi$;
- $\mathsf{Log}, \Pi, i \models \mathsf{tt}$;
- $\mathsf{Log}, \Pi, i \not\models \mathsf{ff}$;
- $\mathsf{Log}, \Pi, i \models \neg\psi$ if $\mathsf{Log}, \Pi, i \not\models \psi$;
- $\mathsf{Log}, \Pi, i \models \psi_1 \wedge \psi_2$ if both $\mathsf{Log}, \Pi, i \models \psi_1$ and $\mathsf{Log}, \Pi, i \models \psi_2$;
- $\mathsf{Log}, \Pi, i \models \psi_1 \vee \psi_2$ if either $\mathsf{Log}, \Pi, i \models \psi_1$ or $\mathsf{Log}, \Pi, i \models \psi_2$;
- $\mathsf{Log}, \Pi, i \models \langle \rho \rangle \psi$ if there exists $j \geq i$ such that $(i, j) \in \mathcal{R}_\Pi(\rho)$ and $\mathsf{Log}, \Pi, j \models \psi$;
- $\mathsf{Log}, \Pi, i \models [\rho]\psi$ if for each $j \geq i$ such that $(i, j) \in \mathcal{R}_\Pi(\rho)$ it holds that $\mathsf{Log}, \Pi, j \models \psi$,

where the relation $\mathcal{R}_\Pi(\rho)$ contains all pairs of indexes (i, j) so that $\Pi(\pi)[i, j]$ conforms to ρ for each π that appears as an index of a proposition in ρ, defined as follows:

- $(i, j) \in \mathcal{R}_\Pi(p_\pi)$ if $j = i + 1$, $\Pi(\pi)[i, j] \neq \epsilon$, and $p \in \Pi(\pi)[i]$;
- $(i, j) \in \mathcal{R}_\Pi(\neg p_\pi)$ if $j = i + 1$, $\Pi(\pi)[i, j] \neq \epsilon$, and $p \notin \Pi(\pi)[i]$;

- $(i,j) \in \mathcal{R}_\Pi(\psi_1 \wedge \psi_2)$ if $(i,j) \in \mathcal{R}_\Pi(\psi_1)$ and $(i,j) \in \mathcal{R}_\Pi(\psi_2)$;
- $(i,j) \in \mathcal{R}_\Pi(\psi_1 \vee \psi_2)$ if $(i,j) \in \mathcal{R}_\Pi(\psi_1)$ or $(i,j) \in \mathcal{R}_\Pi(\psi_2)$;
- $(i,j) \in \mathcal{R}_\Pi(\psi?)$ if $j = i$ and $\mathsf{Log}, \Pi, j \models \psi$;
- $(i,j) \in \mathcal{R}_\Pi(\rho_1 + \rho_2)$ if either $(i,j) \in \mathcal{R}_\Pi(\rho_1)$ or $(i,j) \in \mathcal{R}_\Pi(\rho_2)$;
- $(i,j) \in \mathcal{R}_\Pi(\rho_1;\rho_2)$ if there exists $k \in [i,j]$ such that $(i,k) \in \mathcal{R}_\Pi(\rho_1)$ and $(k,j) \in \mathcal{R}_\Pi(\rho_2)$;
- $(i,j) \in \mathcal{R}_\Pi(\rho^*)$ if $j = i$ or there exists $k \in [i+1,j]$ such that $(i,k) \in \mathcal{R}_\Pi(\rho)$ and $(k,j) \in \mathcal{R}_\Pi(\rho^*)$.

By $\mathsf{Log}, \Pi \models \varphi$ we denote the fact that $\mathsf{Log}, \Pi, 0 \models \varphi$.

A variable π in φ is *free* if it occurs out of the scope of either $\exists\pi$ or $\forall\pi$. By $\mathsf{free}(\varphi)$ we denote the set of free variables in φ. Formula φ is *closed* if $\mathsf{free}(\varphi) = \emptyset$. Observe that the semantics of a closed formula φ replaces all the traces in the assignment Π, which then becomes irrelevant. Thus, we can denote by $\mathsf{Log} \models \varphi$ the fact that $\mathsf{Log}, \Pi, 0 \models \varphi$ for some trace assignment Π.

Note how the semantics allows to consider the evolution of multiple traces at once, in a *synchronous* fashion. Indeed, $\mathcal{R}_\Pi(\rho)$ includes all the pairs (i,j) so that the synchronous execution described by $\Pi(\pi)[i,j]$ conforms to ρ, for each π appearing in the regular expression. As a result, while evaluating formulas of the form $\langle\rho\rangle\psi$ or $[\rho]\psi$, all the traces in Log that are assigned through Π to trace variables appearing in ρ are synchronously executed.

Similarly to HyperLDL$_f$, we can define the hyper-version of LTL$_f$ [De Giacomo and Vardi, 2013].

Definition 2 (HyperLTL$_f$ syntax)**.** *For a set* AP *containing atomic propositions* p *and a set* V *of trace variables* π*, the syntax of HyperLTL$_f$ is the following:*

$$\varphi := \psi \mid \exists\pi\varphi \mid \forall\pi\varphi$$
$$\psi := p_\pi \mid \neg\psi \mid \psi \wedge \psi \mid \psi \vee \psi \mid \mathtt{X}\psi \mid \tilde{\mathtt{X}}\psi \mid \psi\mathtt{U}\psi \mid \mathtt{F}\psi \mid \mathtt{G}\psi$$

In fact, HyperLDL$_f$ can express the classic linear-temporal logic operators over finite traces such as *next* (X) and *until* (U) and so to have HyperLTL$_f$ defined as a syntactic fragment of HyperLDL$_f$. [1] Indeed, the *next* and *weak next* operators can be encoded in HyperLDL$_f$ by $\mathtt{X}\psi \doteq \langle\mathsf{true}_{\mathsf{free}(\psi)}\rangle\psi$ and $\tilde{\mathtt{X}}\psi \doteq \neg\mathtt{X}\neg\psi$, respectively. Note that the propositional true in the diamond operator refers only to the traces that are free in ψ. That is, we do not impose any condition on the rest of the traces, which may or may not terminate (or may have terminated already) at the current execution step. Analogously, for the *until* operator, we have that $\psi_1\mathtt{U}\psi_2 \doteq \langle(\psi_1?\,;\mathsf{true}_{\mathsf{free}(\psi_2)})^*\rangle\psi_2$. Note that in this case it is sufficient to impose the propositional true only on the traces occurring in ψ_2 since, for those occurring in ψ_1, this is automatically imposed by evaluating ψ_1 at each step of the execution specified in the regular expression of the diamond operator. Clearly, we can define the *finally* and *globally* operators as usual as $\mathtt{F}\psi \doteq \mathsf{true}_{\mathsf{free}(\psi)}\mathtt{U}\psi$ and $\mathtt{G}\psi \doteq \neg\mathtt{F}\neg\psi$. These shortcut definitions for the temporal operators automatically induce the semantics of HyperLTL$_f$.

[1] In fact HyperLTL$_f$ is a proper fragment of HyperLDL$_f$, since LTL$_f$ can only express star-free regular expressions [De Giacomo and Vardi, 2013].

Note that, we can impose two traces to be of equal length by $|\pi_1| = |\pi_2| \doteq \mathtt{G}(\mathsf{last}_{\pi_1} \leftrightarrow \mathsf{last}_{\pi_2})$. Analogously, we can say that π_1 is not longer than π_2 by $|\pi_1| \leq |\pi_2| \doteq (\neg\mathsf{last}_{\pi_2})\mathtt{U}\mathsf{last}_{\pi_1}$. Finally, we can also compare the content of traces. For example, we can say that π_1 and π_2 evaluate a subset $L \subseteq$ AP in the same way: $equal_L(\pi_1,\pi_2) \doteq |\pi_1| = |\pi_2| \wedge \mathtt{G}\bigwedge_{p\in L}(p_{\pi_1} \leftrightarrow p_{\pi_2})$.

4 Model Checking HyperLDL$_f$

We are interested in model checking HyperLDL$_f$ properties over a log. In particular we focus on a quite general class of logs: those that are *regular*, i.e., the set of traces in the log can be recognized by a *deterministic finite automaton*, which we call the *Log DFA* $\mathcal{D}$, i.e., $\mathsf{Log} = \mathcal{L}(\mathcal{D})$. We denote $\mathcal{D}, \Pi, i \models \varphi$ the fact that $\mathcal{L}(\mathcal{D}), \Pi, i \models \varphi$, and by $\mathcal{D} \models \varphi$ the fact that $\mathcal{D}, \Pi, 0 \models \varphi$ for some trace assignment Π. Therefore, model checking in our context is defined as follows.

Definition 3. *For a given Log DFA* $\mathcal{D}$ *and a closed HyperLDL$_f$ formula* φ*, the* model-checking *problem is the problem of deciding whether* $\mathcal{D} \models \varphi$*.*

We solve the model-checking problem by reducing it to the emptiness problem of a suitably defined regular automaton. Consider a quantifier free HyperLDL$_f$ formula ψ and let $V = \{\pi_1,\ldots,\pi_n\} = \mathsf{free}(\psi)$. Thus, ψ can be regarded as an LDL$_f$ formula over $\bigcup_{\pi\in V}\mathrm{AP}_\pi$, where AP_π denotes the alphabet AP relativized with variable π, i.e., $\mathrm{AP}_\pi = \{p_\pi : p \in \mathrm{AP}\}$. However, we need to consider its semantics in the context of HyperLDL$_f$, which provides, at once, traces of different length. In order to use the LDL$_f$ automata construction, we restrict to trace assignments that map every variable in traces all of the same length. To do this, consider the function tail mapping a trace assignment Π into another $\mathsf{tail}(\Pi)$ over the same set V of traces, in which each $\Pi(\pi)$ is replaced with $\mathsf{tail}(\Pi)(\pi) = \Pi(\pi) \cdot \mathsf{end}_\pi^{\mathsf{length}(\Pi)-|\Pi(\pi)|+1}$ [2]. Intuitively, every trace mapped in Π is appended with a (repeated) tail of symbols end_π in a way that each of them has the same length.

By $\Theta_\pi = 2^{\mathrm{AP}_\pi}$ we denote the set of possible truth-assignments over the set AP_π and by $\Theta_\pi^{\mathsf{end}} = \Theta_\pi \cup \{\mathsf{end}_\pi\}$, the set of assignments augmented with the corresponding end_π symbol. Observe that the symbol end_π differs from the empty assignment, setting every propositional variable false, but rather mean that the trace is *finished*. Indeed, it will hold that both $\mathsf{end}_\pi \not\models \langle p_\pi\rangle\mathtt{tt}$ and $\mathsf{end}_\pi \not\models \langle\neg p_\pi\rangle\mathtt{tt}$.

By $\Theta_V^{\mathsf{end}} = \bigcup_{\pi\in V}\Theta_\pi^{\mathsf{end}}$ we denote the union set of assignments over the atomic propositions typed with $\pi \in V$. Note that the sets of truth-assignments are pairwise disjoint. Therefore, a truth-assignment in Θ_V^{end} can be uniquely identified with a list of assignments in $\Theta_\pi^{\mathsf{end}}$, one for each $\pi \in V$. At this point, every trace assignment $\mathsf{tail}(\Pi)$ of this form can be regarded as a single trace over the set Θ_V^{end}, to which a quantifier-free HyperLDL$_f$ formula can be interpreted as an LDL$_f$ one. We have the following proposition.

Proposition 1. *For a given quantifier-free HyperLDL$_f$ formula* φ *and a trace assignment* Π*, it holds that* $\Pi \models \varphi$ *if, and*

[2] By $\mathsf{length}(\Pi) = \max_{\pi\in V}\{|\Pi(\pi)|\}$ we denote the length of the longest trace mapped by Π.

only if, $\mathsf{tail}(\Pi) \models \varphi$. [3]

Hence, without loss of generality, we assume the trace assignments to map traces of the same length and then to regard them as a single trace over Θ_V^{end}. This allows us to apply the DFA construction for LDL$_f$ [De Giacomo and Vardi, 2013; Brafman *et al.*, 2018]:

Proposition 2. *For every quantifier free HyperLDL$_f$ formula ψ there exists an* alternating finite automaton $\mathcal{A}_\psi$ *of size polynomial in $|\psi|$ such that, for every trace assignment Π, it holds that $\Pi \models \psi$ if, and only if, $\Pi \in \mathcal{L}(\mathcal{A}_\psi)$.*

Next we focus on regular logs. First, recall that the set Log of valid traces over AP is described in terms of a DFA $\mathcal{D} = \langle \Sigma, Q, q_0, \delta, F \rangle$ with $\Sigma = 2^{\mathrm{AP}}$, that is $\mathsf{Log} = \mathcal{L}(\mathcal{D})$. In our model-checking technique, we may need to read traces that are tailed with the end symbol a finite but unbounded number of times. This is done by amending the automaton and defining $\mathcal{D}^{\mathsf{tail}} = \langle \Sigma', Q', q_0, \delta', F' \rangle$, where $\Sigma' = \Sigma \cup \{\mathtt{end}\}$, $Q' = Q \cup \{\mathtt{acc}, \mathtt{rej}\}$, $F' = F \cup \{\mathtt{acc}\}$ and δ' extends δ as follows:

- $\delta'(q, \mathtt{end}) = \begin{cases} \mathtt{acc} & \text{if } q \in F \\ \mathtt{rej} & \text{otherwise} \end{cases}$;
- $\delta'(\mathtt{acc}, \sigma) = \begin{cases} \mathtt{acc}, & \text{if } \sigma = \mathtt{end} \\ \mathtt{rej} & \text{otherwise} \end{cases}$;
- $\delta'(\mathtt{rej}, \sigma) = \mathtt{rej}$.

Intuitively, $\mathcal{D}^{\mathsf{tail}}$ is as $\mathcal{D}$ until the end of trace is reached. From that point on, it accepts only if both the trace is accepted by $\mathcal{D}$ and the rest is made by an arbitrarily long sequence of end symbols. The following trivially holds.

Proposition 3. *For every DFA $\mathcal{D}$, the automaton $\mathcal{D}^{\mathsf{tail}}$ is such that $\mathcal{L}(\mathcal{D}^{\mathsf{tail}}) = \{w; \mathsf{end}^n : w \in \mathcal{L}(\mathcal{D}) \text{ and } n \in \mathbb{N}\}$.*

Finally we deal with the quantifiers of HyperLDL$_f$. To do so we make use of the classic notions of existential and universal *projection* (see e.g. [David E. Muller and Paul E. Schupp, 1995]).

Definition 4 (Automata projection)**.** *For a given automaton $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F \rangle$ such that $\Sigma = 2^X$ and $P \subseteq X$:*

- *If $\mathcal{A}$ is nondeterministic, then the* existential projection *of $\mathcal{A}$ over P is the automaton $\mathcal{A}_P^\exists = \langle \Sigma, Q, q_0, \delta_P^\exists, F \rangle$ with $\delta_P^\exists(q, \sigma) = \bigvee_{\sigma'_P \in 2^P} \delta(q, \sigma_{X \setminus P} \cup \sigma'_P)$* [4]
- *If $\mathcal{A}$ is universal, then the* universal projection *of $\mathcal{A}$ over P is the automaton $\mathcal{A}_P^\forall = \langle \Sigma, Q, q_0, \delta_P^\forall, F \rangle$ where $\delta_P^\forall(q, \sigma) = \bigwedge_{\sigma'_P \in 2^P} \delta(q, \sigma_{X \setminus P} \cup \sigma'_P)$*

Observe that, by using the Ndet and Univ, we can compute the existential and universal projections of any kind of automaton $\mathcal{A}$. However this might require an exponential blow-up in its size, as nondeterminization and universalization are involved. For simplicity, we use the notation $\mathcal{A}_P^\exists$, in place of $(\mathsf{Ndet}(\mathcal{A}))_P^\exists$, to denote the existential projection for every automaton. Analogously, $\mathcal{A}_P^\forall$ denotes the universal projection for every automaton. Intuitively, the existential projection $\mathcal{A}_P^\exists$ accepts words w for which there exists a replacement of the variables in P that make the resulting word w' accepted by $\mathcal{A}$. The universal projection, instead, $\mathcal{A}_P^\forall$ accepts words w for which every possible replacement of the variable evaluation in P is a word w' accepted by $\mathcal{A}$. This is formalized in the following theorem which adapts a key result in [David E. Muller and Paul E. Schupp, 1995] for infinite words to finite ones.

Theorem 1. *For a given automaton $\mathcal{A}$ over the alphabet 2^X and a subset $P \subseteq X$, the two following propositions hold:*

1. *For every word $w \in (2^X)^*$, $w \in \mathcal{L}(\mathcal{A}_P^\exists)$ iff there exists a word w' s.t. $|w'| = |w|$, $w'(i)_{X \setminus P} = w(i)_{X \setminus P}$ for every $i < |w|$, and $w' \in \mathcal{L}(\mathcal{A})$.*
2. *For every word $w \in (2^X)^*$, $w \in \mathcal{L}(\mathcal{A}_P^\forall)$ iff for every word w' such that $|w'| = |w|$ and $w'(i)_{X \setminus P} = w(i)_{X \setminus P}$ for every $i < |w|$, it holds that $w' \in \mathcal{L}(\mathcal{A})$.*

For a given DFA $\mathcal{D}$ reading words over an alphabet $\Sigma = 2^{\mathrm{AP}}$, the π-typification of $\mathcal{D}$, denoted $\mathcal{D}_\pi$ is obtained by renaming every proposition p in AP to p_π. By $\mathrm{AP}_\pi = \{p_\pi : p \in \mathrm{AP}\}$ we denote the set of typed variables from AP.

We are now ready to present our algorithm that, given a Log DFA $\mathcal{D}$ and a HyperLDL$_f$ formula φ, constructs an automaton $\mathsf{Automaton}(\mathcal{D}, \varphi)$ whose nonemptiness problem is equivalent to check whether $\mathcal{D} \models \varphi$. We first define Algorithm 1 for generating the automaton $\mathsf{Automaton}(\mathcal{D}, \varphi)$.

Algorithm 1 Automata construction $\mathsf{Automaton}(\cdot, \cdot)$.

Input: HyperLDL$_f$ formula φ and Log DFA $\mathcal{D}$.
Output: The automaton $\mathsf{Automaton}(\mathcal{D}, \varphi)$.

if φ *is quantifier free* **then**
  return $\mathcal{A}_\varphi \cap \bigcap_{\pi \in \mathsf{free}(\varphi)} \mathcal{D}_\pi^{\mathsf{tail}}$
if $\varphi = \exists \pi \varphi'$ **then**
  return $(\mathsf{Automaton}(\mathcal{D}, \varphi'))_{\mathrm{AP}_\pi}^\exists$
if $\varphi = \forall \pi \varphi'$ **then**
  return $(\mathsf{Automaton}(\mathcal{D}, \varphi'))_{\mathrm{AP}_\pi}^\forall$

Algorithm 1 recursively scans the formula on a top-to-bottom fashion, handling its quantifiers one by one. At each recursive call, it removes the outermost quantifier of φ and computes the automata projection that corresponds to the quantification modality. For an existential quantification it applies an existential projection, whereas for a universal quantification, it applies a universal one. Algorithm 1 makes use of a polynomial operation: the *typification* of the alphabet.

Theorem 2. *For every HyperLDL$_f$ formula φ and a Log DFA $\mathcal{D}$ it holds that $\mathcal{L}(\mathsf{Automaton}(\mathcal{D}, \varphi)) = \{\Pi : V \to \mathcal{L}(\mathcal{D}) : \mathcal{D}, \Pi, 0 \models \varphi\}$.*

Proof sketch. By induction on the structure of φ, the base case handles the quantifier free formulas, for which the algorithm returns the DFA construction of $\mathcal{A}_\varphi$ intersected with a suitable number of copies of $\mathcal{D}_{\mathrm{AP}_\pi}^{\mathsf{tail}}$ to make sure the selected traces are in the log. The inductive cases handle the quantifiers by means of automata projection. The correctness of such procedure follows from Theorem 1. □

[3] Note that, being φ quantifier free, the set Log becomes irrelevant for its semantic, as it just needs to include the traces mapped to in Π. For this reason, we can omit it.

[4] Note that σ is a subset of X and that for some $Y \subseteq X$, by σ_Y we classically denote the projection over Y, that is $\sigma \cap Y$.

We can now solve model-checking of $\mathcal{D}$ and φ by checking the nonemptiness of $\mathsf{Automaton}(\mathcal{D}, \varphi)$, see Algorithm 2.

Algorithm 2 Model checking HyperLDL$_f$ over Log DFAs.

Input: HyperLDL$_f$ formula φ and Log DFA $\mathcal{D}$.
Output: YES, if $\mathcal{D} \models \varphi$. NO, otherwise.

Compute $\mathcal{A} = \mathsf{Automaton}(\mathcal{D}, \varphi)$ by **Alg. 1**
if $\mathcal{L}(\mathcal{A}) \neq \emptyset$ **then**
  return YES
else
  return NO

The correctness of Algorithm 2 is a direct consequence of Theorem 2. Algorithm 2 gives us an upper-bound on the computational complexity of the problem.

Theorem 3. *The model checking problem for a HyperLDL$_f$ formula φ with quantifier alternation depth k over a Log DFA $\mathcal{D}$ can be solved in k-*EXPSPACE *in both φ and $\mathcal{D}$.*

Proof sketch. For quantified free formulas φ, Algorithm 2 checks the nonemptiness of a polynomial sized alternating automaton, whose complexity is PSPACE. For all the other cases, at each automata projection, the size of the automaton increases by one exponential, thus requiring an extra exponent in space complexity to check for its nonemptiness. □

This bound is tight since we prove a matching lower-bound for the model-checking of both HyperLDL$_f$ and HyperLTL$_f$.

Theorem 4. *The model checking problem for a HyperLTL$_f$ (and so HyperLDL$_f$) formula φ with quantifier alternation depth k over a Log DFA $\mathcal{D}$ is k-*EXPSPACE-Hard.

Proof sketch. Through an encoding from Turing machines. The proof adapts the argument for proving the lower-bound of the satisfiability problem for QPTL in [Sistla *et al.*, 1987]. □

We observe that our technique is based on automata constructions involving projection (existential projection) and determinization and complementation (universal projection, considering that $\forall\varphi \equiv \neg\exists\neg\varphi$). Note that these operations are computationally dominated by determinization (projection is polynomial as is complementation after deteminization). In fact the alternation of quantifiers is directly related to the number of required determinization steps (after projection). This is what makes the tower of exponential directly related to quantifier alternations. On the other hand there is evidence that determinization in the case of regular nondeterministic automata, while worst case exponential, is often polynomial in practice for minimized DFAs [Tabakov and Vardi, 2005] and this often gives rise to scalable algorithms in spite of the worst case computational complexity [Zhu *et al.*, 2017; Zhu *et al.*, 2020; Tabajara and Vardi, 2020]. For this reason Algorithm 1 and 2 are indeed promising also in practice.

5 HyperLDL$_f$ and BPM

We illustrate how HyperLDL$_f$ can be used to capture a variety of interesting properties within Business Process Management (BPM). An operational process consists of a collection of activities that are executed in coordination [Dumas *et al.*, 2018]. The process is instantiated multiple times, and each instance represents the execution of the process on a specific, so-called case object (an order, a claim, ...), moving it from the initial state of the process to one of its final states. This happens through the application of finitely many activities to the case object, in agreement with the ordering constraints imposed by the process control-flow. This setting perfectly matches HyperLDL$_f$, thanks to the ability of the logic to express hyper-properties over the finite traces induced by multiple process instances.

We fix a set A of activities. Given a trace π over A, we assume that at each step, at most one activity is executed. The case where no activity is executed, which we represent for convenience using formula $nop_\pi = \bigwedge_{\mathsf{a}\in A} \neg \mathsf{a}_\pi$, represents an idle step in the execution. With this notions at hand, we concentrate on three relevant aspects in the BPM spectrum.

Process Mining is a family of techniques used to get insights about operational processes, and to improve processes based on event data obtained from actual executions [van der Aalst and others, 2011]. Process mining techniques range from the discovery of process models from event data to checking whether an event log conforms to a process model (either manually crafted or automatically discovered from data). To do so, they need to inspect and relate different traces that are contained in the event log or that can be generated by the process model (two notions that in our setting homogeneously map to the HyperLDL$_f$ notion of regular log). Specifically, several process mining techniques do not directly work with raw traces, but rather use *all* the traces at hand to establish behavioral relations between activities, in turn using them as a succinct, abstract representation.

As a representative example, we consider here the reference *ordering relations* introduced in one of the seminal works on process discovery [van der Aalst *et al.*, 2004]. Such ordering relations capture whether, according to the log, two activities directly follow each other, are mutually exclusive, or concurrent. For example, a log Log indicates that activities a and b directly follow each other, that is, are in a *strict sequencing* (written $\mathsf{a} \rightarrow_{\mathsf{Log}} \mathsf{b}$ in [van der Aalst *et al.*, 2004]), if: *(i)* there exists a trace in Log where a is followed by b without any other activity in between; *(ii)* in none of the traces from Log the converse happens, that is, b is followed by a without any other activity in between. While these two properties separately quantify over traces, they have to holds simultaneously in Log to infer that a and b are in a strict sequencing, hence calling for the hyper-features of HyperLDL$_f$. Indeed, we can check whether $\mathsf{a} \rightarrow_{\mathsf{Log}} \mathsf{b}$ by:

$$\mathsf{Log} \models \exists\pi_1 \forall\pi_2 \; \langle(\mathsf{true}_{\pi_1})^*; \mathsf{a}_{\pi_1}; (nop_{\pi_1})^*; \mathsf{b}_{\pi_1}\rangle \mathsf{tt} \wedge [(\mathsf{true}_{\pi_2})^*; \mathsf{b}_{\pi_2}; (nop_{\pi_2})^*; \mathsf{a}_{\pi_2}]\mathsf{ff}$$

The other ordering relations can be formalized analogously.

Instance-Spanning Constraints are behavioral constraints that span across multiple instances of a given process, and thus inherently require to related multiple traces at once. In [Rinderle-Ma *et al.*, 2016], a multitude of instance-spanning constraints is collected from different application scenarios, from logistics to healthcare and security. Interestingly, previous formalization attempts in this spectrum either failed

or required to combine all distinct traces in a single stream equipping events with trace identifiers [Rinderle-Ma *et al.*, 2016], an approach that is not applicable when the (possibly infinite) log to be considered is generated by a process model. Differently from those attempts, thanks to its hyper-features HyperLDL$_f$ has the direct ability to elegantly capture instance-spanning constraints.

Consider for example an *activity propagation* constraint dictating that whenever an activity a occurs in a trace π of the log, then b has to occur *later on* in all traces π' *related to* π. By assuming that the fact that π is related to π' is expressed through a domain-specific HyperLDL$_f$ formula $rel(\pi_1, \pi_2)$ having π and π' as free variables, we can formalize in HyperLDL$_f$ this activity propagation constraint as: $\forall \pi_1, \pi_2\, rel(\pi_1, \pi_2) \rightarrow \mathtt{G}(\mathsf{a}_{\pi_1} \rightarrow \mathtt{XF}\, \mathsf{b}_{\pi_2})$. This formula can be instantiated to capture that whenever a package shipped via truck is detected as contaminated, then all the packages traveling in the same truck need to be later inspected.

Alternative behaviors in a process. We finally consider the use of HyperLDL$_f$ to express sophisticated properties on alternative behaviors exhibited by a process that cannot be captured using branching-time logics. To illustrate this, we first partition the activities A into *observable activities* A_{O} (public activities executed by or visible to external actors), and *unobservable activities* A_{U} (private activities). We want to express a *variety of outcome* property capturing that whenever the process can produce a trace π_1 satisfying a desired behavioral pattern (a regular expression ρ) and leading to a given outcome (a final, observable activity), then it is also able to produce a different trace π_2 that *(i) mimics* π_1 by exhibiting the same *observable behavior*, and *(ii)* satisfies the *same* desired pattern ρ leading to an *alternative* outcome.

Intuitively, by "mimicking" we mean that two traces are equal once all unobservable activities are projected out. Additionally, observable activities are not required to happen at the same time: an observable activity happening in either trace is replicated by the other with an arbitrary delay by adding unobservable activities in between. Until this happens, however, no other observable activity can be executed in either trace. Note that the role of mimicking and of being mimicked can be exchanged among π_1 and π_2 at each observable activity: it can happen that an observable activity of π_1 may be mimicked by π_2 with a certain delay, and that the next observable activity of π_2 is then mimicked by π_1 with some other delay. This requirement cannot be captured by a simple branching-time formula but is indeed a hyper property that requires the ability to compare observable projections of traces.

This can be used, for example, to check that the process is so that whenever a customer properly follows a sequence of public steps required by a payment system, if the payment is ultimately rejected then there is an alternative execution, with the same public steps but with possibly different unobservable ones (in type or number), where the payment is accepted.

To formalize this, we assume that in each step at most one task is executed. Let ρ_π be a regular expression over the set A in π, and $\mathsf{a}, \mathsf{b} \in A_{\mathrm{U}}$ respectively denote the actual and alternative outcomes. Then property can be captured as:

$$\forall \pi_1 \exists \pi_2 (\langle \rho_{\pi_1} \rangle \mathsf{a} \rightarrow (\langle \rho_{\pi_2} \rangle \mathsf{b} \wedge mimics(\pi_1, \pi_2)))$$

where $mimics(\pi_1, \pi_2)$ formalizes that π_2 mimics π_1. Since π_1 and π_2 may have a different length, we write $mimics(\pi_1, \pi_2) = |\pi_1| \leq |\pi_2| \wedge mimicsObs(\pi_1, \pi_2) \vee |\pi_1| > |\pi_2| \wedge mimicsObs(\pi_2, \pi_1)$, and then formalize this notion of mimicking via $mimicsObs(\pi_s, \pi_l)$, taking advantage from the fact that π_s is of equal or shorter length than π_l. To do so, we define a support formula: given a set T of trace variables, $un_T = \bigwedge_{\pi \in T}(nop_\pi \vee (\bigvee_{u \in A_{\mathrm{U}}} u_\pi))$ captures an unobservable step jointly performed in all traces from T. Using this, we write $\varphi_{skip} = (un_{\{\pi_s, \pi_l\}})^*$ to indicate an arbitrary sequence of possibly different unobservable steps simultaneously performed by π_s and π_l. As a result, $mimicsObs(\pi_s, \pi_l) =$

$$\left\langle \left(\bigvee_{\mathsf{p} \in A_{\mathrm{O}}} \begin{pmatrix} (\varphi_{skip}; \mathsf{p}_{\pi_s} \wedge \mathsf{p}_{\pi_l}) \\ \vee (\varphi_{skip}; \mathsf{p}_{\pi_s} \wedge un_{\{\pi_l\}}; \varphi_{skip}; \mathsf{p}_{\pi_l} \wedge un_{\{\pi_s\}}) \\ \vee (\varphi_{skip}; \mathsf{p}_{\pi_l} \wedge un_{\{\pi_s\}}; \varphi_{skip}; \mathsf{p}_{\pi_s} \wedge un_{\{\pi_l\}}) \end{pmatrix}^* \right) \right\rangle$$
$$\langle \varphi_{skip} \rangle (\mathsf{finished}_{\pi_s} \wedge \langle (un_{\{\pi_l\}})^* \rangle \mathsf{finished}_{\pi_l})$$

6 Related Work on Infinite Traces

To deal with multiple traces at once, a specific extension of LTL, called HyperLTL, has been originally introduced in the infinite traces setting. HyperLTL supports explicit quantification over traces, as well as hyper-properties that inspect and relate multiple traces at once [Clarkson *et al.*, 2014]. In [Clarkson *et al.*, 2014], HyperLTL and HyperCTL* are introduced as extensions of LTL and CTL* to reason about a set of (infinite) traces all at once. A hyper extension of PDL, namely HyperPDL-Δ, is considered in [Gutsfeld *et al.*, 2020]. The model-checking and satisfiability problems for HyperLTL and these extensions have been investigated and solved [Clarkson *et al.*, 2014; Finkbeiner and Hahn, 2016; Gutsfeld *et al.*, 2020; Mascle and Zimmermann, 2020]. Their solutions are based on automata on infinite objects (traces), and provide a similar complexity hierarchy as the one in Theorem 3. Our technique avoids the manipulation of automata on infinite objects, sidestepping the difficulties of complementation of such automata, in favor of standard automata for regular languages, which have better computational characteristics in practice, see the discussion at the end of Section 4.

7 Conclusion

We have introduced HyperLDL$_f$ a logic for expressing hyper properties over logs, which are of particular interest, e.g., in BPM. Specifically, we have proposed a technique for model-checking of regular logs that takes full advantage of the relationship between LTL$_f$/LDL$_f$ and regular automata. For this reason, such a technique promises to be easily implementable and effective in practice in spite of the high worst-case complexity of the problem. It is also of interest to study other problems such as synthesis of a log satisfying certain HyperLDL$_f$ properties, which is related to satisfiability, or synthesis of controller generating a log with desired HyperLDL$_f$ properties, cfr. [Pnueli and Rosner, 1989].

Acknowledgments Research partially supported by the ERC Advanced Grant WhiteMech (No. 834228) and by the EU ICT-48 2020 project TAILOR (No. 952215).

References

[Baier and McIlraith, 2006] Jorge A. Baier and Sheila A. McIlraith. Planning with temporally extended goals using heuristic search. In *ICAPS*, pages 342–345. AAAI, 2006.

[Baier *et al.*, 2007] Jorge A. Baier, Christian Fritz, and Sheila A. McIlraith. Exploiting procedural domain control knowledge in state-of-the-art planners. In *ICAPS*, pages 26–33. AAAI, 2007.

[Brafman *et al.*, 2018] Ronen I. Brafman, Giuseppe De Giacomo, and Fabio Patrizi. Ltlf/ldlf non-markovian rewards. In *AAAI*, pages 1771–1778. AAAI Press, 2018.

[Chandra *et al.*, 1981] Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.

[Clarkson *et al.*, 2014] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In *ETAPS 2014*, pages 265–284, 2014.

[David E. Muller and Paul E. Schupp, 1995] David E. Muller and Paul E. Schupp. Simulating Alternating Tree Automata by Nondeterministic Automata: New Results and New Proofs of the Theorems of Rabin, McNaughton and Safra. *Theor. Comput. Sci.*, 141(1&2):69–107, 1995.

[Davis *et al.*, 1994] Martin D. Davis, Ron Sigal, and Elaine J. Weyuker. *Computability, Complexity, and Languages (2nd Ed.): Fundamentals of Theoretical Computer Science*. Academic Press Professional, Inc., USA, 1994.

[De Giacomo and Vardi, 2013] Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*, pages 854–860, 2013.

[De Giacomo *et al.*, 2014] Giuseppe De Giacomo, Riccardo De Masellis, Marco Grasso, Fabrizio Maria Maggi, and Marco Montali. Monitoring business metaconstraints based on LTL and LDL for finite traces. In *BPM*, volume 8659 of *LNCS*, pages 1–17. Springer, 2014.

[Dumas *et al.*, 2018] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management, Second Edition*. Springer, 2018.

[Finkbeiner and Hahn, 2016] Bernd Finkbeiner and Christopher Hahn. Deciding hyperproperties. In *CONCUR*, volume 59 of *LIPIcs*, pages 13:1–13:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

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

[Gutsfeld *et al.*, 2020] J. O. Gutsfeld, M. Müller-Olm, and C. Ohrem. Propositional dynamic logic for hyperproperties. In *CONCUR # "20"*, 2020. To appear.

[Leiss, 1985] Ernst L. Leiss. Succinct Representation of Regular Languages by Boolean Automata II. *Theor. Comput. Sci.*, 38:133–136, 1985.

[Mascle and Zimmermann, 2020] Corto Mascle and Martin Zimmermann. The keys to decidable hyperltl satisfiability: Small models or very simple formulas. In *CSL*, volume 152 of *LIPIcs*, pages 29:1–29:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[Pesic *et al.*, 2007] Maja Pesic, Helen Schonenberg, and Wil M. P. van der Aalst. DECLARE: full support for loosely-structured processes. In *EDOC*, pages 287–300. IEEE Computer Society, 2007.

[Pnueli and Rosner, 1989] A. Pnueli and R. Rosner. On the Synthesis of a Reactive Module. In *POPL'89*, pages 179–190. Association for Computing Machinery, 1989.

[Rinderle-Ma *et al.*, 2016] Stefanie Rinderle-Ma, Manuel Gall, Walid Fdhila, Jürgen Mangler, and Conrad Indiono. Collecting examples for instance-spanning constraints. *CoRR*, abs/1603.01523, 2016.

[Sistla *et al.*, 1987] A.P. Sistla, M.Y. Vardi, and P. Wolper. The Complementation Problem for Büchi Automata with Applications to Temporal Logic. *TCS*, 49:217–237, 1987.

[Tabajara and Vardi, 2020] Lucas M. Tabajara and Moshe Y. Vardi. LTLf Synthesis under Partial Observability: From Theory to Practice. In *GandALF*, volume 326 of *EPTCS*, pages 1–17, 2020.

[Tabakov and Vardi, 2005] Deian Tabakov and Moshe Y. Vardi. Experimental evaluation of classical automata constructions. In *LPAR*, volume 3835 of *LNCS*, pages 396–411. Springer, 2005.

[Vaandrager, 2017] Frits W. Vaandrager. Model learning. *Commun. ACM*, 60(2):86–95, 2017.

[van der Aalst and others, 2011] Wil M. P. van der Aalst et al. Process mining manifesto. In *Business Process Management Workshops (1)*, volume 99 of *LNBIP*, pages 169–194. Springer, 2011.

[van der Aalst *et al.*, 2004] Wil M. P. van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. Knowl. Data Eng.*, 16(9):1128–1142, 2004.

[van der Aalst, 2011] Wil M. P. van der Aalst. *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.

[Weidlich *et al.*, 2011] Matthias Weidlich, Artem Polyvyanyy, Nirmit Desai, Jan Mendling, and Mathias Weske. Process compliance analysis based on behavioural profiles. *Inf. Syst.*, 36(7):1009–1025, 2011.

[Zhu *et al.*, 2017] Shufang Zhu, Lucas M. Tabajara, Jianwen Li, Geguang Pu, and Moshe Y. Vardi. Symbolic LTLf synthesis. In Carles Sierra, editor, *IJCAI*, pages 1362–1369. ijcai.org, 2017.

[Zhu *et al.*, 2020] Shufang Zhu, Giuseppe De Giacomo, Geguang Pu, and Moshe Y. Vardi. LTLf synthesis with fairness and stability assumptions. In *AAAI*, pages 3088–3095. AAAI Press, 2020.