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

Giuseppe De Giacomo, Marco Montali, Paolo Felli, Giuseppe Perelli

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\textsubscript{f} and its extension LDL\textsubscript{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 HyperLDLf, a logic that extends LDL\textsubscript{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\textsubscript{f} and LDL\textsubscript{f}. These logics turned out to be particularly important in BPM: LTL\textsubscript{f} has become the formal basis for one of the most studied declarative process modeling notations [Pesic et al., 2007], while LDL\textsubscript{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\textsubscript{f} and LDL\textsubscript{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 HyperLDLf and its fragment HyperLDLf\textsubscript{f}: two extensions of LDL\textsubscript{f} and LTL\textsubscript{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 HyperLDLf formulas over a regular log, i.e., a (possibly infinite) set of traces forming a regular language. We study the HyperLDLf\textsubscript{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 HyperLDLf\textsubscript{f} formula with \(k\) alternations of trace quantifiers is \(k\)-EXSPACE-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\textsubscript{f}/LDLf. Finally, we go back to our original motivation...
and discuss how HyperLTL\textsubscript{f} and HyperLDL\textsubscript{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 exploit later [Davis et al., 1994]. For a given set \( X \), \( B^+(X) \) is the set of positive Boolean formulas over \( X \), that is Boolean formulas built from elements of \( X \) using \( \land \) and \( \lor \), where we also allow true and false. A subset \( Y \subseteq X \) satisfies a Boolean formula \( \chi \in B^+(X) \) if the truth assignment assigning true to the elements in \( Y \) and false to the elements in \( X \setminus Y \) satisfies \( \chi \). Given an alphabet \( \Sigma \), a finite word over it is a finite sequence \( w = \sigma_0 \cdot \sigma_1 \cdots \sigma_n \) of letters in \( \Sigma \). By \( |w| = n + 1 \) we denote the length of \( w \). Moreover, by \( \Sigma^{\ast} \) we denote the set of all possible finite words over \( \Sigma \). An alternating finite automaton (AFA) on finite words is a tuple \( A = (\Sigma, Q, q_0, \delta, F) \) where \( \Sigma \) is the (finite) input alphabet, \( Q \) is a finite set of states with \( q_0 \in Q \) being initial state, \( \delta : Q \times \Sigma \to B^+(Q) \) 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 \( A \) can reach when reading the symbol \( \sigma \) while being in state \( q \). The automaton \( A \) is nondeterministic (NFA), denoted \( N \), or universal (UFA), denoted \( U \), if \( \delta(q, \sigma) \) contains only disjunctions or conjunctions, respectively. Finally, it is deterministic (DFA), denoted \( D \), if the transition function maps each pair \( (q, \sigma) \) to a single state \( q' \).

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

3 HyperLDL\textsubscript{f} over Finite Traces

We start by defining the hyper-version of LDL\textsubscript{f}. We refer to [De Giacomo and Vardi, 2013] for details on LDL\textsubscript{f} itself.

\textbf{Definition 1 (HyperLDL\textsubscript{f} syntax).} For a set \( C \) containing atomic propositions \( p \) and a set \( V \) of trace variables \( \pi \), the syntax of HyperLDL\textsubscript{f} is the following:
\[
\begin{align*}
\varphi & ::= \psi \mid \exists \pi \varphi \mid \forall \pi \varphi \\
\psi & ::= tt \mid ff \mid \neg \psi \mid \psi \land \psi \mid \psi \lor \psi \mid (\rho) \psi \mid [\rho] \psi \\
p & ::= \phi \mid [\rho] \psi \mid p + \rho \mid p / \rho \mid p^* \\
\phi & ::= p_{\pi} \mid -p_{\pi} \mid \phi \land \phi \mid \phi \lor \phi
\end{align*}
\]

Observe that the atomic propositions occurring in a HyperLDL\textsubscript{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 \( \text{true}_p \triangleq \rho_p \lor \neg p_p \) and \( \text{false}_p \triangleq \neg \text{true}_p \) for every trace variable \( p \) occurring in a HyperLDL\textsubscript{f} formula. Analogously, by \( \text{true}_p \triangleq \bigwedge_{\pi \in P} \text{true}_p \) and \( \text{false}_p \triangleq \neg \text{true}_p \) we denote the propositional true and false over a set of traces. For convenience we use the abbreviation finished\( _\pi \triangleq [\text{true}_\pi]^{ff} \) to specify that a trace \( \pi \) is finished, and \( \text{last}_\pi \triangleq (\text{true}_\pi) \text{finished}_\pi \) to specify the last element of the trace.

A log \( \log \subseteq (2^{AP})^* \) 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 \( t \in \log \), \( |t| \) 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 from \( t[i] \) to \( t[j] \). 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) \( \pi : V \to \log \) is a function mapping trace variables \( V \) to concrete traces in \( \log \). By \( \pi[i] \triangleq t \) we denote the trace assignment that adheres with \( \pi \), except that \( \pi[i] \triangleq t(i) = t \), that is, \( t \) is assigned to \( \pi \). By \( \pi[i, j] \), for every \( i, j \in \mathbb{N} \), we denote the projection of assignment \( \pi \) over the substraces between \( i \) and \( j \), i.e., such that \( \pi[i, j] = \pi[i] \) for every \( \pi \in V \). By \( \pi[i, \ldots] \), for every \( i \in \mathbb{N} \), we denote the projection of \( \pi \) over the suffix from \( i \), i.e., such that \( \pi[i, \ldots] = \pi[i] \) for each \( \pi \in V \).

The semantics of a HyperLDL\textsubscript{f} formula over a set Log of traces, a trace assignment \( \pi \), and a natural number \( i \in \mathbb{N} \) is given as follows:
\begin{itemize}
\item Log, \( i \models \exists \pi \varphi \) if there exists \( t \in \log \) such that Log, \( \pi[t] \models \varphi \);
\item Log, \( i \models \forall \pi \varphi \) if for each \( t \in \log \) it holds that Log, \( \pi[t] \models \varphi \);
\item Log, \( i \models tt \);
\item Log, \( i \models ff \);
\item Log, \( i \models \neg \psi \) if Log, \( i \not\models \psi \);
\item Log, \( i \models \psi_1 \land \psi_2 \) if both Log, \( i \models \psi_1 \) and Log, \( i \models \psi_2 \);
\item Log, \( i \models \psi_1 \lor \psi_2 \) if either Log, \( i \models \psi_1 \) or Log, \( i \models \psi_2 \);
\item Log, \( i \models (\rho) \psi \) if there exists \( j \geq i \) such that \( (i, j) \in R(\rho) \) and Log, \( j \models \psi \);
\item Log, \( i \models [\rho] \psi \) if for each \( j \geq i \) such that \( (i, j) \in R(\rho) \) it holds that Log, \( j \models \psi \);
\end{itemize}

where the relation \( R(\rho) \) contains all pairs of indexes \( (i, j) \) so that \( \pi[i, j] \) conforms to \( \rho \) for each \( \rho \) that appears as an index of a proposition in \( \rho \), defined as follows:
\textbf{-} (\textbf{1}) \( (i, j) \in R(\rho(p)) \) if \( j = i + 1, \pi[i, j] \neq \epsilon, \) and \( p \in \Pi(\pi)[i] \);
\textbf{-} (\textbf{2}) \( (i, j) \in R(\neg p) \) if \( j = i + 1, \pi[i, j] \neq \epsilon, \) and \( p \not\in \Pi(\pi)[i] \).
Note that, we can impose two traces to be of equal length by \(|\pi_1| = |\pi_2| \equiv G_{\text{last}^{\pi_1} \leftrightarrow \text{last}^{\pi_2}}\). Analogously, we can say that \(\pi_1\) is not longer than \(\pi_2\) by \(|\pi_1| \leq |\pi_2| \equiv \neg (\text{last}^{\pi_2} \text{last}^{\pi_1})\). Finally, we can also compare the content of traces. For example, we can say that \(\pi_1\) and \(\pi_2\) evaluate a subset \(L \subseteq \text{AP}\) in the same way: \(\text{equal}_L(\pi_1, \pi_2) \equiv |\pi_1| = |\pi_2| \land G \bigwedge_{p \in L} (p_{\pi_1} \leftrightarrow p_{\pi_2})\).

4 Model Checking HyperLDL<sub>f</sub>

We are interested in model checking HyperLDL<sub>f</sub> 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 \(D\), i.e., \(L = L(D)\). We denote \(D, \Pi, i \models \varphi\) the fact that \(L(D), \Pi, i \models \varphi\), and by \(D \models \varphi\) the fact that \(D, \Pi, 0 \models \varphi\) for some trace assignment \(\Pi\). Therefore, model checking in our context is defined as follows.

Definition 3. For a given Log DFA \(D\) and a closed HyperLDL<sub>f</sub> formula \(\varphi\), the model-checking problem is the problem of deciding whether \(D \models \varphi\).

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

By \(\Theta_\pi = 2^{AP_\pi}\) we denote the set of possible truth-assignments over the set \(\text{AP}_\pi\) and by \(\Theta_\pi^{\text{end}} = \Theta_\pi \cup \{\text{end}_\pi\}\), the set of assignments augmented with the corresponding \(\text{end}_\pi\) symbol. Observe that the symbol \(\text{end}_\pi\) differs from the empty assignment, setting every propositional variable false, but rather mean that the trace is \(\text{finished}\). Indeed, it will hold that both \(\text{end}_\pi \neq (p_\pi)^{tt}\) and \(\text{end}_\pi \neq (\neg p_\pi)^{tt}\).

By \(\Theta_\pi^{\text{end}} = \bigcup_{\pi \in V} \Theta_\pi^{\text{end}}\) 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 \(\Theta_\pi^{\text{end}}\) can be uniquely identified with a list of assignments in \(\Theta_\pi^{\text{end}}\), one for each \(\pi \in V\).

At this point, every trace assignment \(\Pi(\text{tail})\) of this form can be regarded as a single trace over the set \(\Theta_\pi^{\text{end}}\), to which a quantifier-free HyperLDL<sub>f</sub> formula can be interpreted as an LDL<sub>f</sub> one. We have the following proposition.

Proposition 1. For a given quantifier-free HyperLDL<sub>f</sub> formula \(\varphi\) and a trace assignment \(\Pi\), it holds that \(\Pi \models \varphi\) if, and
Proposition 2. For every quantifier free HyperLDL formula \( \psi \) there exists an alternating finite automaton \( A_\psi \) of size polynomial in \(|\psi|\) such that, for every trace assignment \( \Pi \), it holds that \( \Pi \models \psi \) if, and only if, \( \Pi \in L(A_\psi) \).

Next we focus on regular logs. First, recall that the set \( \text{Log} \) of valid traces over \( Y \) we classically denote the projection over \( \pi \). Then, for every automaton \( D \), by Proposition 3. For every DFA \( D \) reading words over an alphabet \( \Sigma = 2^X \), the \( \pi \)-typification of \( D \), denoted \( D_\pi \), is obtained by renaming every proposition \( p \) in \( \text{AP} \) by \( p_\pi \). By \( \text{AP}_\pi = \{ p_\pi : p \in \text{AP} \} \) we denote the set of typed variables from \( \text{AP} \).

Finally we deal with the quantifiers of HyperLDL. 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 \( A = (\Sigma, Q, q_0, \delta, F) \) such that \( \Sigma = 2^X \) and \( P \subseteq X \):
- If \( A \) is nondeterministic, then the existential projection of \( A \) over \( P \) is the automaton \( A^\exists_P = (\Sigma, Q, q_0, \delta^\exists_P, F) \) with \( \delta^\exists_P(q, \sigma) = \bigvee_{P' \in 2^P} \delta(q, \sigma_{X \setminus P} \cup \sigma_{P'}) \).
- If \( A \) is universal, then the universal projection of \( A \) over \( P \) is the automaton \( A^\forall_P = (\Sigma, Q, q_0, \delta^\forall_P, F) \) where \( \delta^\forall_P(q, \sigma) = \bigwedge_{P' \in 2^P} \delta(q, \sigma_{X \setminus P} \cup \sigma_{P'}) \).

Observe that, by using the Ndet and Univ, we can compute the existential and universal projections of any kind of automaton \( A \). However this might require an exponential blow-up in its size, as nondeterminization and universalization are involved. For simplicity, we use the notation \( A^\exists_P \), in place of \( (\text{Ndet}(A))^\exists_P \), to denote the existential projection for every automaton. Analogously, \( A^\forall_P \) denotes the universal projection for every automaton. Intuitively, the existential projection \( A^\exists_P \) accepts words \( w \) for which there exists a replacement of the variables in \( P \) that make the resulting word \( w' \) accepted by \( A \). The universal projection, instead, \( A^\forall_P \) accepts words \( w \) for which every possible replacement of the variable evaluation in \( P \) is a word \( w' \) accepted by \( A \). 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 \( A \) over the alphabet \( 2^X \) and a subset \( P \subseteq X \), the two following propositions hold:
1. For every word \( w \in (2^X)^* \), \( w \in L(A^\exists_P) \) 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 \( w' \in L(A) \).
2. For every word \( w \in (2^X)^* \), \( w \in L(A^\forall_P) \) 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 \( w' \in L(A) \).

Algorithm 1 Automata construction Automaton(\( \cdot, \cdot \)).

Input: HyperLDL formula \( \varphi \) and Log DFA \( D \).
Output: The automaton Automaton(\( D, \varphi \)).

\[
\text{if } \varphi \text{ is quantifier free then} \\
\quad \text{return } A_\varphi \cap \bigcap_{p \in \text{free}(\varphi)} D^\exists_p \\
\text{if } \varphi = \exists p \varphi' \text{ then} \\
\quad \text{return } \left( \text{Automaton}(D, \varphi') \right)^\exists_{\text{AP}_\pi} \\
\text{if } \varphi = \forall p \varphi' \text{ then} \\
\quad \text{return } \left( \text{Automaton}(D, \varphi') \right)^\forall_{\text{AP}_\pi}
\]

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 \( \varphi \) and computes the automata projection that corresponds to the quantification modalities. 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 formula \( \varphi \) and a Log DFA \( D \) it holds that \( L(\text{Automaton}(D, \varphi)) = \{ \Pi : V \rightarrow L(D) : D, \Pi, 0 \models \varphi \} \).

Proof sketch. By induction on the structure of \( \varphi \), the base case handles the quantifier free formulas, for which the algorithm returns the DFA construction of \( A_\varphi \) intersected with a suitable number of copies of \( D^\exists_{\text{AP}_\pi} \) 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.
We can now solve model-checking of $D$ and $\varphi$ by checking the nonemptiness of Automaton($D, \varphi$), see Algorithm 2.

Algorithm 2 Model checking HyperLDL$_f$ over Log DFAs.
Input: HyperLDL$_f$ formula $\varphi$ and Log DFA $D$.
Output: YES, if $D \models \varphi$, NO, otherwise.
Compute $A = \text{Automaton}(D, \varphi)$ by Alg. 1
if $L(A) \neq \emptyset$ 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 $\varphi$ with quantifier alternation depth $k$ over a Log DFA $D$ can be solved in $k$-EXPSPACE in both $\varphi$ and $D$.

Proof sketch. For quantified free formulas $\varphi$. 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 $\varphi$ with quantifier alternation depth $k$ over a Log DFA $D$ 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 determinization). 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 $\pi$ 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 $\text{nop}_a = \bigwedge_{a \in A} \neg a_\pi$, 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 $a \rightarrow_{\text{Log}} b$ 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 $a \rightarrow_{\text{Log}} b$ by:

$$\text{Log} \models \exists \pi_1 \forall \pi_2 \left( (\text{true}_{\pi_1})^*; a_{\pi_1}; (\text{nop}_{\pi_1})^*; b_{\pi_2}; (\text{nop}_{\pi_2})^*; a_{\pi_2} \right)$$

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 hyperfeatures 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 $\pi$ of the log, then $b$ has to occur later on in all traces $\pi'$ related to $\pi$. By assuming that the fact that $\pi$ is related to $\pi'$ is expressed through a domain-specific HyperLDL$_f$ formula $\text{rel}(\pi_1, \pi_2)$ having $\pi$ and $\pi'$ as free variables, we can formalize in HyperLDL$_f$ this activity propagation constraint as: 

$$\forall \pi_1, \pi_2 \text{rel}(\pi_1, \pi_2) \Rightarrow G(a_{\pi_1} \rightarrow XF b_{\pi_2}).$$

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 $\pi_1$ satisfying a desired behavioral pattern (a regular expression $\rho$) and leading to a given outcome (a final, observable activity), then it is also able to produce a different trace $\pi_2$ that (i) mimics $\pi_1$ by exhibiting the same observable behavior, and (ii) satisfies the same desired pattern $\rho$ 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 executed by or visible to external actors, and unobservable activities executed by or visible to external actors), and unobservable activities executed by private actors. We want to express a variety of outcome property capturing that whenever the process can produce a trace $\pi_1$ satisfying a desired behavioral pattern (a regular expression $\rho$) and leading to a given outcome (a final, observable activity), then it is also able to produce a different trace $\pi_2$ that (i) mimics $\pi_1$ by exhibiting the same observable behavior, and (ii) satisfies the same desired pattern $\rho$ leading to an alternative outcome.

$$\forall \pi_1 \exists \pi_2 ((\rho_{\pi_1})a \rightarrow ((\rho_{\pi_2})b \land \text{mimics}(\pi_1, \pi_2)))$$

where $\text{mimics}(\pi_1, \pi_2)$ formalizes that $\pi_2$ mimics $\pi_1$. Since $\pi_1$ and $\pi_2$ may have a different length, we write $\text{mimics}(\pi_1, \pi_2) = |\pi_1| \leq |\pi_2| \land \text{mimicsObs}(\pi_1, \pi_2) \lor |\pi_2| \leq |\pi_1| \land \text{mimicsObs}(\pi_2, \pi_1)$, and then formalize this notion of mimicking via $\text{mimicsObs}(\pi_s, \pi_1)$, taking advantage from the fact that $\pi_s$ is of equal or shorter length than $\pi_1$. To do so, we define a support formula: given a set $T$ of trace variables, $\text{un}_T = \bigwedge_{\pi \in T} (\text{nop}_{\pi} \lor (\bigvee_{u \in A_U} u_{\pi}))$ captures an unobservable step jointly performed in all traces from $T$. Using this, we write $\varphi_{\text{skip}} = (\text{un}_{\pi_1})^* \pi_s \land \text{skip}\varphi_{\text{skip}} \land \pi_1 \land \text{skip}\varphi_{\text{skip}} \land \text{un}_{\pi_1}$ to indicate an arbitrary sequence of possibly different unobservable steps simultaneously performed by $\pi_s$ and $\pi_1$. As a result, $\text{mimicsObs}(\pi_s, \pi_1) = \langle (\varphi_{\text{skip}})^{\nu} \rangle_{\text{skip}} \varphi_{\text{skip}} \land \text{un}_{\pi_1}$

### 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$\Delta$, 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; Mascele 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


