Model Checking Multi-Agent Systems against LDLK Specifications on Finite Traces

Jeremy Kong
Department of Computing, Imperial College London
London, United Kingdom
jeremykong91@gmail.com

Alessio Lomuscio
Department of Computing, Imperial College London
London, United Kingdom
a.lomuscio@imperial.ac.uk

ABSTRACT
We introduce the logic $\text{LDL}_f^K$, a variant of the epistemic logic LDLK, interpreted on finite traces of multi-agent systems. We explore the verification problem of multi-agent systems against $\text{LDL}_f^K$ specifications and give algorithms for the reduction of $\text{LDL}_f^K$ model checking to LDLK verification on a different model and different specification. We analyse the resulting complexity and show it to be PSPACE-complete. We report on a full implementation of the algorithm and assess its performance on a number of examples.

KEYWORDS
Verification; model checking; epistemic logic; LDL

1 INTRODUCTION
Several approaches have been put forward to address the verification of multi-agent systems (MAS). Differently from work in software validation and in reactive systems, MAS are typically specified by using expressive specification languages that go beyond the expressive power used in other areas. MAS specifications are often based on AI-attitudes such as knowledge, beliefs, desires, intentions, strategic abilities, normative states, etc. In particular, methods addressing specifications involving epistemic states of the agents in the system and their strategic ability to bring about states of affairs have been developed [1, 2, 4, 6, 7, 14, 28, 29, 33–36, 38].

Indeed, several toolkits have been developed for model checking MAS against epistemic specifications [21, 27, 37] and strategic ones [3, 12, 37]. While the methods vary, one common aspect is that the treatment of the underlying temporal specifications is based on logics such as LTL or CTL.

However, in other areas of AI, proposals have been made for the adoption of linear dynamic logic (LDL) as an underlying model of time [17, 43] in a variety of applications, over infinite and finite traces. The advantage of LDL over LTL is a considerably increased expressivity (LDL is equivalent to monadic second-order logic), while having the same model checking complexity of LTL. The increased expressivity has been positively evaluated in applications including in planning and services [15]. Contributions have recently been made on the use of LDL in an agent-based context. Specifically, as we report below, a symbolic approach for the verification of MAS against an epistemic extension of LDL has been put forward [30]. However, this approach is limited to LDL over infinite traces. In this paper we surpass this limitation and explore verification of MAS against LDLK specifications over finite traces.

While models where computation paths are infinite have infinite have featured extensively in computer science, recently there has been attention to logics defined on finite executions, particularly in AI. This is because in AI applications such as planning, it is of interest to investigate whether temporal states of affairs hold in finite executions. It is known, however, that the semantics of LTL and LDL on finite traces (LTL$_f$ and LDL$_f$, respectively) requires care [5, 16], suggesting that finite traces should be studied separately.

In this paper we introduce the logic $\text{LDL}_f^K$, an epistemic variant of LDL on finite traces, and put forward an approach for the resulting model checking problem. We study the resulting complexity and show that the model checking problem remains PSPACE-complete. In this way we obtain an expressive logic whose complexity is no higher than that of LTL. In addition to theoretical results, we present practical algorithms for verifying MAS against LDLK specifications. We also report on an implementation of these into a prototype toolkit, built from the open-source model checker MCMAS [37], and report on its performance on several applications.

Related Work. [24] presented a semantics for LTL$_f$ and an algorithm for verifying LTL$_f$ specifications in the OBJ specification language [22]. The work presented here supports a different semantics, based on [17], and a more expressive range of specifications both in terms of the temporal and epistemic component.

LDL$_f$ was introduced in [17], which also presented an algorithm for converting an LDL$_f$ model checking problem into a nonemptiness test for a suitably constructed alternating automaton. Our approach is different as we do not solve the LDL$_f$ model checking problem directly, instead, we reduce it to LDL model checking. Moreover, we also address epistemic properties of the agents.

LTL$_f$ and LDL$_f$ have also been used as temporal logics for specifying temporally extended goals (TEGs) in planning. [42] proposes an approach for deterministic domains which compiles LDL$_f$ formulae into alternating automata, and encodes them into the domain. [10] addresses strong cyclic planning for LTL$_f$ TEGs in non-deterministic domains via a different compilation into automata. More generally, [18, 19] discuss synthesis for LTL$_f$ and LDL$_f$. Our work instead focuses on MAS verification.

LDL$_f$ also sees application in multi-agent contexts. [23] considers strategy synthesis for iterated Boolean games with LDL$_f$ goals. As above, our work instead focuses on verification, not synthesis. Also, agents in [23] also have perfect information, while in our treatment agents only have imperfect information.

Related to this work is also [30], where the logic LDLK was put forward in the context of MAS, and the model checking problem was originally shown to be PSPACE-complete. Differently from the approach here taken, the semantics given there is on infinite traces; this has different properties from those here analysed.

The semantics we adopt for LDLK in terms of path termination bears resemblance to approaches in module checking as discussed in [32]. This has been extended to multi-agent contexts in [26]. Our work differs in that specifying when and how paths are terminated we report the experimental results obtained and comment on the performance of the tool. We conclude in Section 6.

2 THE LOGIC LDLK ON FINITE TRACES

Interpreted systems [20] are a semantics for reasoning about temporal and epistemic properties of MAS. In the variant we use here, from [37], each agent has a locally defined transition relation; composing these yields the global transition relation of the system. We define \( A = \{1, \ldots, n\} \) as a set of agents and \( E \) as a special agent called the environment.

Definition 1. An interpreted system may be expressed as a tuple \( IS = (\langle L_i, Act_i, P_i, \tau_i \rangle_{i \in A \cup \{E\}}, I, h) \), where:

- \( L_i \) is a finite set of possible local states of agent \( i \).
- \( Act_i \) is a finite set of possible actions of agent \( i \).
- \( P_i : L_i \rightarrow 2^{Act_i} \setminus \{\emptyset\} \) is a local protocol for agent \( i \), specifying which actions agent \( i \) can execute from each local state.
- \( \tau_i : L_i \times Act_i \times \ldots \times Act_{t_i} \times Act_E \rightarrow L_i \) is a local transition function, returning the next local state of agent \( i \) following an action by all agents and the environment. We assume agents’ actions’ sequences are consistent with their protocols; i.e., if \( \tau_i(l_i, a_1, \ldots, a_n, a_E) \), then for each \( i \in \{1, \ldots, n\} \), we have that \( a_i \in P_i(l_i) \).
- \( I \subseteq L_1 \times \ldots \times L_n \times L_E \) is the set of initial global states.
- \( h : AP \rightarrow 2^{L_1 \times \ldots \times L_n \times L_E} \), where \( AP \) is a set of atomic propositions, is a valuation function identifying the global states in which each atomic proposition holds.

We define a global transition relation \( T \), the composition of the \( \tau_i \), and the set of global reachable states \( G \subseteq L_1 \times \ldots \times L_n \times L_E \) as the states reachable through zero or more applications of \( T \) on \( I \).

We also allow referencing individual states on a path. We denote \( \pi(k) \) to be the \( k \)-th state of a path \( \pi = \pi_0, \pi_1, \ldots, \pi_k \) to be the \( k \)-th suffix of \( \pi \), i.e., \( \pi_k = \pi_{k-1}, \ldots, \pi_0 \).

Finally, for each agent \( i \in A \) we also define a projection \( l_i : G \rightarrow L_i \). This returns the local state of agent \( i \) in a given global state.

Syntax of LDLK on Finite Traces. The syntax of LDLK on finite traces, hereafter LDLK, is identical to that of regular LDLK, as introduced in [30].

Definition 2. An LDLK formula \( \phi \) is constructed as follows:

\[
\langle \phi \rangle ::= \top | \neg \phi | \phi \land \psi | K_i \phi | E_r \phi | D_r \psi | C_r \psi | \langle \rho \rangle \phi
\]

\( p \) is a propositional atom; \( i \in A \cup \{E\} \) is an agent, \( \Gamma \subseteq A \cup \{E\} \) a set of agents, and \( \psi \) a formula not containing LDL operators \( \langle \rho \rangle \phi \) not in scope of an epistemic operator. We allow standard propositional abbreviations, and a dual for the modal diamond operator \( [\rho] \phi \equiv \neg \langle \neg \rho \rangle \neg \phi \).

Intuitively, \( \rho \) may be viewed as a path expression that a given prefix of a path may satisfy. The reading of \( \langle \rho \rangle \phi \) is that there exists some path prefix that satisfies \( \rho \), after which \( \phi \) is true. The reading of \( [\rho] \phi \) is that after every path prefix satisfying \( \rho \), \( \phi \) is true.

Note that throughout the paper we use observational semantics for the agents’ knowledge.

While the syntax of LDLK is identical to that of LDLK, its intuitive meaning is different, particularly when dynamic modalities are nested. We illustrate these differences by discussing various specifications in the context of the train gate controller scenario from [25] with just two trains.

- \( \langle \top \rangle \langle \text{tun}_1 \rangle \). The reading is the same whether the formula is interpreted on finite or on infinite traces ("Train 1 eventually enters the tunnel"). However, on finite traces train 1 must enter the tunnel before the trace ends.
- \( \langle \top \rangle (K_c \langle \text{tun}_1 \rangle \rightarrow \langle \top \rangle (\langle \top \rangle \langle \text{tun}_2 \rangle)) \). On infinite traces the reading is "When the controller knows that train 1 is in the tunnel, then train 2 will be in the tunnel some positive even number of steps later". On finite traces, the interpretation is similar, but it also implies that the controller must not know that train 1 is in the tunnel in the last two states of any trace.
- \( \langle \top \rangle (\langle \top \rangle \langle \text{tun}_1 \rangle) \). On infinite traces the reading is "Train 1 enters the tunnel infinitely often". However, on finite traces this is read as "train 1 is in the tunnel at the end of a trace".
- \( \langle \top \rangle (\langle \top \rangle \langle \text{tun}_1 \rangle) \). On infinite traces this is read as "Train 1 enters the tunnel infinitely often on even states". On finite traces, however, this instead means "train 1 is in the tunnel on the last even state of the trace".
- \( \langle \neg K_c (\langle \text{tun}_1 \rangle \land \langle \text{tun}_2 \rangle) \rangle \). \( K_c \langle \top \rangle \langle \text{tun}_1 \rangle \) is true. This formula is read in the same way for both finite and infinite traces ("while train 2 does not know that train 1 will eventually enter the tunnel, it remains inside; once it knows this, it leaves the tunnel, though it may return later").
- \( K_c (\langle \text{tun}_1 \rangle \land \langle \text{tun}_2 \rangle) \). Note that this formula is unsatisfiable on infinite traces, since \( \langle \top \rangle \langle \text{tun}_1 \rangle \) cannot be satisfied; it therefore is equivalent to falsity. However, on finite traces \( \langle \top \rangle \langle \text{tun}_1 \rangle \) can hold if the trace has ended, since no prefix will match \( \top \). Thus, on finite traces, the formula expresses that

\footnote{For example, \( K_a (p) \land \neg q \) is allowed for \( \psi \), but \( K_a (p) \land q \) is not.} \footnote{Consider the prefix that consists of all but the last state of a given trace.}
“the controller knows that the trains enter the tunnel in alternation for the duration of the trace, and the run ends with train 2 in the tunnel”. Note that in general, [T]ₐ allows us to characterise the end of a trace [17].

Given the above we conclude that LDLᵦ is replete to render very expressive specifications over finite traces. We also observe that the meaning of LDLᵦ is very different from that of syntactically equivalent LDL₉ formulae, suggesting that the expressivity of LDLᵦ should be explored separately from LDL₉.

Having discussed the differences in the readings of formulas in finite and infinite traces we now introduce satisfaction. Satisfaction for LDLᵦ is interpreted over finite traces.

**Definition 3.** Let IS = ⟨(L, Act, Pᵦ, τᵦ)ᵦ∈E⟩, I, h⟩ be an interpreted system and let π be an infinite path induced by IS. Furthermore, let Last be a non-negative integer that indicates the index of the last state of π to be considered. Then, the satisfaction of specification φ on the interpreted system IS over path π ending at Last (formally: IS, π, Last ⊨ φ) is inductively defined as follows:

- IS, π, Last ⊨ p if and only if h(p).
- IS, π, Last ⊨ ⊤.
- IS, π, Last ⊨ φ₁ ∧ φ₂ if it is the case that both IS, π, Last ⊨ φ₁ and IS, π, Last ⊨ φ₂.
- IS, π, Last ⊨ (p)φ if there exists 0 ≤ i ≤ Last such that (0, i) ∈ R(π, π₁) and IS, π₁, (Last − i) ⊨ φ. Note that the latter term is well-defined, since i ≤ Last.
- IS, π, Last ⊨ Kφ if for every state g ∈ G if l₁(π₁) = l₁(π₀), then IS, g ⊨ φ.
- IS, π, Last ⊨ Eᵦφ if for every state g ∈ G if l₁(π₁) = l₁(π₀) for some agent i ∈ Γ then IS, g ⊨ φ.
- IS, π, Last ⊨ Dᵦφ if for every state g ∈ G if l₁(π₁) = l₁(π₀) for all agents i ∈ Γ then IS, g ⊨ φ.
- IS, π, Last ⊨ Cᵦφ if for every state g ∈ G if g₁, g₂, . . . are states such that l₁(π₁) = l₁(π₀), l₁(π₂) = l₁(π₀), . . . , l₁(πₙ) = l₁(π₀) for some agents i, j, k₁, . . . ∈ Γ and positive integer n, then IS, g ⊨ φ.

The relation R ⊆ (p × π) × (N × N) is defined very similarly to LDLK, though we need to account for the finiteness of traces:

- R(φ, π) = {(i, i + 1) : IS, π₁, Last − i |= φ}
- R(φ₁, φ₂) = {(i, i) : IS, π₁, Last − i |= φ₂}
- R(p + p’, π) = R(p, s) ∪ R(p’, s)
- R(p(p’, π) = {(i, j) : ∃k s.t. (i, k) ∈ R(p, π) ∧ (k, j) ∈ R(p’, π)}
- R(p’, π) = {(i, j) : ∃k s.t. (i, k) ∈ R(p, π) ∧ (k, j) ∈ R(p’, π)}

Observe that above there is a semantic difference between LDL operators and epistemic ones. While LDL operators are interpreted on the finite path up to Last, the epistemic possibilities for the knowledge operators are defined on all global states of the model. This is in marked difference with approaches such as bounded model checking for epistemic logic whereby only a fraction of the model is considered both for temporal and epistemic modalities [41]. The reason for this is that the semantics here presented is not intended as a verification method. It captures, instead, the intuitive formulation of time and knowledge on finite traces where crucially the length of the path is not commonly known by the agents in the system and may even depend on factors outside their control. As a detailed example of this, we refer to the Go-Back-N protocol discussed in Section 5, where permanent channel failures could bring the protocol to termination; agents may have no foreknowledge of this. As an example from the literature, in [39], an application of LTL specifications on finite traces in the context of monitoring business constraints, the authors state “traces are finite and subject to extensions as new events happen”. The idea that traces may remain of interest for finite but unknown length is also echoed in [15], which considers LDL₉ specifications in this context as well. It is in the spirit of LTL and LDL₉ for the traces to be finite but the trace’s length to be unknown to the agents. Considering all epistemic alternatives as possible precisely captures this aspect.

Intuitively, the path expressions p are regular expressions over propositional or epistemic formulas ψ (with support for tests, which check that a given LDL₉ formula is true in a given state). These follow the standard PDL semantics regarding choice, composition and the Kleene star. (p)φ is satisfied if and only if there exists some prefix of a path matching p, after which φ holds. Conversely, [p]φ means that after every prefix of a path matching p, φ must hold.

As in LDLK, for a global state g ∈ G, we say IS, g ⊨ φ if on every path π starting at g and every value of Last we have IS, π, Last ⊨ φ.

Finally, we introduce the LDLᵦ model checking problem: given an interpreted system IS, state g and LDLᵦ formula φ, determine whether IS, g ⊨ φ.

### 3 MODEL CHECKING LDLᵦ

In this section we introduce a model checking problem for LDLᵦ and give tight bounds for the complexity of the problem. We solve the LDLᵦ model checking problem by reducing it to the LDLK model checking problem (over infinite traces); this can then be solved via the algorithm presented in [30]. Intuitively, this is achieved by modelling when an infinite path should cease to be considered active (since every finite path is a prefix of some infinite path), and translating LDLᵦ formulae to suitable LDLK formulae respecting when paths become inactive.

Formally, given an interpreted system IS, state g in IS and LDLᵦ formula φ, we seek to find another interpreted system IS’, state g’ in IS’ and LDLK formula φ’, such that IS, g ⊨ LDLᵦ, φ if and only if IS’, g’ ⊨ LDLK, φ’.

Note that the expression on the left uses LDLᵦ semantics, whilst that on the right uses LDLK semantics; note also that the models and the formulas are different. We now proceed to define constructively the elements on the right hand side.

**Determining IS’.** To obtain IS’ we add an agent whose purpose is to keep track if the current path remains “active” or not.

**Definition 4.** A **path terminator** P is an agent with local (private) states L_P = {alive, dead} and actions Act_P = {continue, stop}. P has the protocol P_P(alive) = {continue, stop}; P_P(dead) = {stop}. The evolution of P is such that P is in the local state alive after a continue action, and in the local state dead after a stop action.

To define IS’, we also introduce the atomic proposition Alive, holding if and only if the path terminator is in the alive state.

**Definition 5.** Given an IS = ⟨(Lᵦ, Actᵦ, Pᵦ, τᵦ)ᵦ∈E⟩, I, h⟩, we define IS’ = ⟨(Lᵦ, Actᵦ, Pᵦ, τᵦ)ᵦ∈E∪{E, P}, IS’, IS’⟩, where P is as in Definition 4, IS’ is the set of states where the projection of agents’...
states ignoring $P$ matches a state in $I$ and $P$ is alive, and $h'$ is $h$ extended by the $\text{Alice}$ proposition and where we assume that $\text{Alice}$ holds at the initial states.

Observe that the definition above ensures we have paths of length at least 1. We also define a notion of a corresponding path, which links finite traces in $IS$ with infinite paths in $IS'$.  

**Definition 6.** Given an $IS = \{I_!, A!t_1, P_1, t_1 \}_{t_1 \in \mathcal{E}(E)} I, h \}$, $IS'$ as defined in Definition 5, $\pi$ a path in $IS$ and $\text{Last}$ a nonnegative integer, the corresponding path $\pi'$ is the path in $IS'$ such that for any nonnegative integer $n$, we have:

- for agent $i \in A \cup \{E\}$, $l_i(\pi'(n)) = l_i(\pi(n))$;
- for agent $P_i$, $l_P(\pi'(n)) = \begin{cases} \text{Alice} & n \leq \text{Last} \\ \text{Not Alivel} & n > \text{Last} \end{cases}$

Notice that since we have set the states of all of the agents in every possible state, there is only one such path for each $\pi$.

**Determining $\phi'$.** We now present the details leading to the definition of $\phi'$ to account for the semantics of the path terminator. Firstly, we need to ensure that we only consider finite paths. To do this, we need to restrict the paths considered to those where the path terminator is eventually dead. Secondly, the dynamic modalities need to account for the possibility that a path has terminated early. These modalities are satisfied based on statements concerning some or all prefixes of a path conforming to a certain path expression $\rho$; it would not be appropriate to consider prefixes that would not actually be realised if the path terminates before the prefix is completed. We want the path to still be alive after we have matched this sequence. This is accomplished by the following translation algorithm, which uses two mutually recursive procedures and the $\text{Alice}$ predicate defined above.

**Algorithm 1 Finite Restriction**

**INPUT:** LDL$_f$K formula $\phi$

**OUTPUT:** LDLK formula $\phi'$

1. function $\text{Finite-Restrict}(\phi)$
2. return $(\langle \top \rangle)^{(\neg \text{Alice})} \to \text{Live-Restrict}(\phi)$
3. end function

Algorithm 1 encodes the fact that a path must be finite (so the path terminator eventually becomes dead); Algorithm 2 propagates the application of restrictions to subformulae and handles the requirement that for $\langle \rho \rangle \phi$, we only consider paths that are still live after $\rho$ has been satisfied.

We also assume the existence of a function $\text{Translate-Alphabet}$, which applies $\text{Live-Restrict}$ on the formulae in any occurrences of $\psi$ or $\phi'$. This can be easily implemented, e.g., via recursive descent. Notice that the separation of $\text{Finite-Restrict}$ and $\text{Live-Restrict}$ is not strictly necessary for correctness (one could apply $\text{Finite-Restrict}$ throughout). However, we expect that using only $\text{Finite-Restrict}$ would significantly increase the formula size, which LDLK model checking algorithms are sensitive to [30].

We now prove the translation is correct; that is, the answers to the two model checking problems always yield the same results. We first prove several lemmas.

**Algorithm 2 Path Liveness Restriction**

**INPUT:** LDL$_f$K formula $\phi$

**OUTPUT:** LDLK formula $\phi'$

1. function $\text{Live-Restrict}(\phi)$
2. if $\phi$ is an atomic proposition $\rho$ then return $\rho$
3. else if $\phi = \top$ then return $\top$
4. else if $\phi = \neg \phi_1$ then return $\neg \text{Live-Restrict}(\phi_1)$
5. else if $\phi = \phi_1 \land \phi_2$ then return $\text{Live-Restrict}(\phi_1) \land \text{Live-Restrict}(\phi_2)$
6. else if $\phi = K_0 \phi_1$ then return $K_0 \text{Live-Restrict}(\phi_1)$
7. else if $\phi = E_1 \phi_1$ then return $E_1 \text{Live-Restrict}(\phi_1)$
8. else if $\phi = D_1 \phi_1$ then return $D_1 \text{Live-Restrict}(\phi_1)$
9. else if $\phi = C_1 \phi_1$ then return $C_1 \text{Live-Restrict}(\phi_1)$
10. else if $\phi = \text{Translate-Alphabet}(\rho)$ then $\phi = (\rho)\phi_1$
11. return $(\rho')((\text{Alice} \land \text{Live-Restrict}(\phi_1)))$
12. end if
13. end function

**Lemma 3.1.** Let $IS$ be an interpreted system, $\pi$ a path in $IS$, Last a nonnegative integer and $\phi$ an LDL$_f$K formula. Let $IS'$ be $IS$ defined as in Definition 5, and $\pi'$ defined as in Definition 6. Then, $IS', \pi' \models \text{Finite-Restrict}(\phi)$ iff $IS', \pi' \models \text{Live-Restrict}(\phi)$

**Proof.** By definition of $\pi'$, $P$ satisfies $\neg \text{Alice}$ after Last + 1 states. Thus, $IS', \pi' \models (\langle \top \rangle)^{(\neg \text{Alice})}$. Now, consider that $IS', \pi' \models \text{Finite-Restrict}(\phi)$

$\Rightarrow IS', \pi' \models (\langle \top \rangle)^{(\neg \text{Alice})} \to \text{Live-Restrict}(\phi)$

$\Rightarrow IS', \pi' \models \text{Live-Restrict}(\phi)$

**Lemma 3.2.** Let $IS$ be an interpreted system and $\phi$ an LDL$_f$K formula. Let $IS'$ be $IS$ defined as in Definition 5. Let $\pi'$ be a path in $IS'$ and suppose $\pi'$ is not a corresponding path of any path in $IS$. Then, $IS', \pi' \models \text{Finite-Restrict}(\phi)$.

**Proof.** We first show that $\pi'$ must have $\text{Alice}$ permanently true. Suppose this is not true, for a contradiction. Then there must be a first state where $\text{Alice}$ is false, with index $n \geq 1$; we can choose Last = $n$ - 1. We can recover $\pi'$ by taking a projection of $\pi'$ ignoring the path terminator's state. Then $\pi'$ is the corresponding path for $\pi$ and Last = $n$ - 1. Then, by LDLK semantics, $IS', \pi'$ does not satisfy $(\langle \top \rangle)^{(\neg \text{Alice})}$, so $IS', \pi' \models \text{Finite-Restrict}(\phi)$.

**Lemma 3.3.** Let $IS$ be an interpreted system and $\phi$ an LDL$_f$K formula. Let $IS'$ be $IS$ defined as in Definition 5. Let $\pi$ be a path in $IS$, and $\pi'$ the corresponding path in $IS'$. Let $P_{K_a}$ be the set of $\pi'$ in $IS$ and values of Last such that $I_{P_{K_a}}(\pi'(0)) = \phi(0)$, and let $Q_{K_a}$ be the set of paths $\pi'$ in $IS'$ with $I_{P_{K_a}}(\pi'(0)) = \phi(\pi'(0))$. Consider that $Q_{K_a}$ can be partitioned into paths that are corresponding paths from some $\pi''$, and paths that are not. The corresponding partition of $Q_{K_a}$ is precisely the set of corresponding paths to paths in $P_{K_a}, P_{K_a}'$.

**Proof.** Observe that every corresponding path $\pi''$ is $P_{K_a}'$, has $P_{K_a}(\pi''(0)) = I_{Q_{K_a}}(\pi'(0))$ and is in $Q_{K_a}$. Notice that these are equal to $I_{P_{K_a}}(\pi(0))$ and $I_{P_{K_a}}(\pi'(0))$ by construction, and these are equal by definition. Hence $P_{K_a} \subseteq Q_{K_a}$. Conversely, suppose for a contradiction that there is some path $\pi''$ in $Q_{K_a}$ that is a corresponding
path of some $\pi^*\notin \mathbb{P}_K$. This means that $l_0(\pi^*(0)) \neq l_0(\pi(0))$, a contradiction since $l_0(\pi^*(0)) = l_0(\pi(0))$. Thus, $Q_K \subseteq \mathbb{P}_K$.

Thus, the sets are equal.

\[\Box\]

Lemma 3.4. Let $IS$ be an interpreted system, $\pi$ be a path in $IS$, Last be a nonnegative integer and $\phi$ be an $\text{LDL}_K$ formula. Then,

$$IS, \pi, \text{Last} \models \text{LDL}_K \phi \iff IS', \pi' \models \text{LDL}_K \phi'$$

where $IS'$ is defined above; $\phi' = \text{Finite-Restrict}(\phi)$; and $\pi'$ is defined as in Definition 6.

Proof. We proceed by structural induction on $\phi$. The base cases of atomic propositions and $\top$ follow by construction of $IS'$ and $\pi'$.

Notice that by construction of $\pi'$ and $IS'$, we can apply Lemma 3.1 in all cases. It thus suffices to show

$$IS, \pi, \text{Last} \models \text{LDL}_K \phi \iff IS', \pi' \models \text{LDL}_K \text{Live-Restrict}(\phi)$$

The Boolean cases are trivial. For the epistemic modalities, we prove by the inductive hypothesis on $\phi$. By Definition 3, the semantics for satisfaction on states, this holds iff for every path $\pi'$ in $IS$ such that $l_0(\pi'(0)) = l_0(\pi(0))$ and value of Last, $IS, \pi', \text{Last} \models \text{LDL}_K \phi$.

We claim our previous assertion holds for every path $\pi'$ in $IS'$ with $l_0(\pi'(0)) = l_0(\pi(0))$, we have $IS', \pi' \models \text{Finite-Restrict}(\phi)$.

Using Lemma 3.2, the non-corresponding paths must satisfy $\text{Finite-Restrict}(\phi)$. We can thus freely add or drop them. Thus we only need concern ourselves with corresponding paths. By Lemma 3.3 these paths are precisely the corresponding paths of paths in $\pi^*$. Going forward, for each $\pi'$ and value of $\text{Last}$, the corresponding path $\pi'^*$ satisfies $\text{Finite-Restrict}(\phi)$ by the inductive hypothesis. Going backward, if $IS, \pi' \models \text{Finite-Restrict}(\phi)$ and corresponds to path $\pi^*$ and integer $\text{Last} IS, \pi', \text{Last} \models \phi$ by the inductive hypothesis.

Finally, by definition of satisfaction on states we have our assertion holding iff for every state $g'$ in $IS'$ with $l_0(\phi') = l_0(\pi'(0))$, $IS', \pi' \models \text{Finite-Restrict}(\phi)$. This precisely matches the semantics of $IS', \pi' \models K_\phi(\text{Finite-Restrict}(\phi))$. The proofs for $E_\pi$, $D_\pi$ and $C_\pi$ are similar. This completes the proof for the epistemic cases.

Finally, consider the $(\rho)\phi$ case: we have that

$$IS', \pi' \models \text{LDL}_K \text{Live-Restrict}(\rho) \phi$$

$$IS', \pi' \models \text{LDL}_K (\rho') \phi$$

$$\exists \beta \leq i, (0, i) \in \mathcal{R}(\rho', \pi') \land IS, \pi'' \models \text{LDL}_K (\phi \land \text{Live-Restrict}(\phi''))$$

$$\exists \beta \leq i, (0, i) \in \mathcal{R}(\rho', \pi') \land IS, \pi'' \models \text{LDL}_K \text{Live-Restrict}(\phi'')$$

$$\exists \beta \leq i, (0, i) \in \mathcal{R}(\rho', \pi') \land IS, \pi'' \models \text{LDL}_K \text{Live-Restrict}(\phi'')$$

$$\exists \beta \leq i, (0, i) \in \mathcal{R}(\rho', \pi') \land IS, \pi'' \models \text{LDL}_K \text{Live-Restrict}(\phi'')$$

The second last step holds because $\pi'$ is Alive for $\text{Last} + 1$ steps, by construction; the inductive hypothesis is applied in the last step.

Next, we show that $\mathcal{R}_\text{LDL}_K(\rho, \pi) = \mathcal{R}_\text{LDL}_K(\rho', \pi')$ for arbitrary $\rho$, where $\rho' = \text{Translate-Alphabet}(\rho)$. We proceed by structural induction on $\rho$. We first consider the base cases $\mathcal{R}_\text{LDL}_K(\psi, \pi)$ and $\mathcal{R}_\text{LDL}_K(\phi, \pi)$. Notice that these are defined as $\{(i, i + 1) \in IS, \pi', \text{Last} - i \models \phi\}$ and $\{(i, i) : IS, \pi, \text{Last} - i \models \phi\}$ respectively. Since $\psi$ and $\phi$ here are subformulas of $\phi$, we apply the (main) inductive hypothesis. We can thus rewrite the sets as $\{(i, i + 1) : IS', \pi'' \models \text{Finite-Restrict}(\psi)\}$ and $\{(i, i) : IS', \pi'' \models \text{Finite-Restrict}(\phi')\}$ respectively. We then convert instances of $\text{Finite-Restrict}$ to $\text{Translate-Alphabet}$ by Lemma 3.1 and definition of $\text{Translate-Alphabet}$. We then have $\mathcal{R}_\text{LDL}_K(\text{Translate-Alphabet}(\psi), \pi')$ and $\mathcal{R}_\text{LDL}_K(\text{Translate-Alphabet}(\phi), \pi')$, by $\text{LDL}$ semantics. For the inductive cases, notice that the semantics for satisfaction on choice, composition and the Kleene star are identical and only depend on $\mathcal{R}$. Thus, establishing the two base cases is sufficient.

We thus can equate $\mathcal{R}(\rho', \pi')$ with $\mathcal{R}(\rho, \pi)$. This yields

$$\exists \beta \leq i, (0, i) \in \mathcal{R}(\rho, \pi) \land IS, \pi', \text{Last} - i \models \text{LDL}_K \phi$$

and from Definition 3, we have $IS, \pi, \text{Last} \models (\rho)\phi$.

We can now show the main result required.

Theorem 3.5. Let $IS$ be an interpreted system and $g$ be a state in $IS$. Then, $IS, g \models \text{LDL}_K \phi$ if and only if $IS', g' \models \text{LDL}_K \phi'$, where $\phi' = \text{Finite-Restrict}(\phi)$ and $g'$ is $g$ augmented with the path terminator in the state alive.

Proof. By Definition 3, $IS', g' \models \text{LDL}_K \phi'$ if and only if for every path $\pi'$ with $\pi'(0) = g'$, $IS', \pi' \models \text{LDL}_K \text{Finite-Restrict}(\phi)$. By the definition of $\text{Finite-Restrict}$, we have $IS', \pi' \models \text{LDL}_K \text{Finite-Restrict}(\phi)$ for any $\pi'$ where $P$ remains alive.

Thus, we have $IS', g' \models \text{LDL}_K \phi'$ if and only if for every path $\pi'$ where the path terminator is eventually dead, we have $IS', \pi' \models \text{LDL}_K \text{Finite-Restrict}(\phi)$. Such paths must have the path terminator alive for precisely $Last + 1$ steps for some $Last$, and by Lemma 3.4 hold if and only if $IS, \pi, \text{Last} \models \text{LDL}_K \phi$. Since $\pi'$ was arbitrary apart from the eventual death restriction, we have $IS', g' \models \text{LDL}_K \phi'$ if for every path $\pi$ and value of $Last$ we have $IS, \pi, \text{Last} \models \text{LDL}_K \phi$, i.e. if and only if $IS, g \models \text{LDL}_K \phi$.

Computational Complexity. We now turn to explore the complexity of model checking interpreted systems against $\text{LDL}_K$ specifications. We assume, as usual, that interpreted systems are given explicitly. We first bound the size of $IS'$ and $\phi'$:

Lemma 3.6. Let $IS$ be an interpreted system and $IS'$ be defined as above. Let $IS'$ be $IS$ extended with a path terminator as defined above. Then, the size of $IS'$, $|IS'|$ is polynomially bounded in $|IS|$.

Proof. Clearly, $|G'| \leq 2|G|$ as each state in $IS$ can be mapped to at most two states in $IS'$, the corresponding states in which the path terminator is alive or dead. The transition relation can be bounded by the square of the size of the state space (e.g. via an adjacency matrix). Hence, $|IS'| \leq 4|IS|$ (asymptotically).

Lemma 3.7. Let $\phi$ be an $\text{LDL}_K$ formula. Let $\phi' = \text{Finite-Restrict}(\phi)$. Then, the size of $\phi'$, $|\phi'|$ is polynomially bounded in $|\phi|$.

Proof. Consider that $\phi'$ is generated by Algorithm 1. Consider the formation tree of $\phi$. Each node in the tree experiences at most one direct application of Algorithm 1 and at most one direct application of Algorithm 2. Each of these algorithms, apart from recursive calls on subformulas, adds a constant number of literals and connectives. Thus, these add at most a constant factor to the space required to represent that subformula. Hence, $|\phi'|$ is at most a constant factor larger than $|\phi|$, and thus it is polynomially bounded.

The inequality is strict if there are some states in $I$ that can never be revisited.
We can then show the following tight result.

**Theorem 3.8.** Model checking interpreted systems against LDL\(_f\)K specifications is PSPACE-complete.

**Proof.** We first show hardness. Since an LDL\(_f\) formula is an LDL\(_f\)K formula, and LDL\(_f\) model checking is PSPACE-complete hence PSPACE-hard [17], LDL\(_f\)K model checking is PSPACE-hard.

We claim that our approach for solving the model checking problem for LDL\(_f\)K can be implemented in polynomial space.

Concretely, IS and IS' can be done in polynomial space. Further, constructing IS' and \(\phi'\) can be done in polynomial space. Consider that an algorithm for constructing IS' can simply duplicate each state in IS and then force the path terminator to be alive in one copy and dead in the other. Building the transition relation could involve replicating the edges in the original transition relation for pairs of corresponding states in IS'.

We can use a similar argument to that in Lemma 3.7 to show that computing Finite-Restrict via Algorithm 1 runs in polynomial time. Each node in the formation tree is visited at most once by Algorithm 1 and at most once by Algorithm 2. The amount of work done at each node is polynomial. Thus, our Finite-Restrict algorithm runs in polynomial time, and thus in polynomial space.

We can thus perform model checking in LDL\(_f\)K by transforming the LDL\(_f\)K model checking problem to an LDLK model checking problem, where using Lemmas 3.6 and 3.7 the size of the new model and formula are polynomial in the size of the original model and formula. Since LDLK itself is solvable in polynomial space [30] and the composition of polynomials is also polynomial, the model checking algorithm for LDL\(_f\)K also runs in polynomial space. □

Also observe that the algorithm is fixed-parameter tractable in that we have exponentiality only in the size of the formula and not in the size of the model, provided the LDLK model checking algorithm is fixed-parameter tractable (such as that presented in [30]).

### 4 IMPLEMENTATION DETAILS

We implemented the algorithms introduced in Section 3 on top of MCMAS\(_{LDLK}\) [30], a publicly available extension of MCMAS 1.3.0 [37]. The source code and binaries for the implementation, MCMAS\(_{LDLK}\), are available from [40].

MCMAS\(_{LDLK}\) accepts MAS descriptions in ISPL [37]. The syntax for the specifications is that of MCMAS\(_{LDLK}\) [30]. Verification over finite traces is carried out by MCMAS\(_{LDLK}\) by invoking the tool with the command-line flag `-ldlf`. This allows for additional efficiency as we only need to perform verification over finite traces. We do not need to separately retain IS and IS' in memory.

Upon invocation, the tool constructs IS' by adding an agent which behaves as the path terminator from Definition 4. This has a local variable encoding whether it is alive. The model is also updated by adding an Evaluation variable for the path terminator being alive, and a constraint in the set of initial states (InitStates in ISPL) imposing the path terminator to begin in the alive state.

The reduction of \(\phi\) to \(\phi'\) is further optimised with respect to what was presented in Section 3.2. Specifically, note that MCMAS offers support for model checking with Fairness (justice) constraints; a fairness constraint \(p\) is true iff \(p\) is true infinitely often on a given path [8]. Given this we add a fairness constraint !Alive so that we do not consider the paths where the path terminator remains alive (i.e., infinite paths). The path terminator cannot resurrect itself once dead, so this also encodes the requirement that paths are finite.

We thus do not need to add this restriction in Finite-Restrict. In our implementation, Finite-Restrict(\(\phi\)) can simply return Live-Restrict(\(\phi\)) as MCMAS is able to use our fairness constraint to consider only finite paths. This is beneficial for performance, as the LDLK model checking algorithm that MCMAS\(_{LDLK}\) uses may take time exponential in the formula size [30].

MCMAS\(_{LDLK}\) also supports counterexample generation; this involves finding a finite trace \(\pi\) in IS where IS, \(\pi \not\models \phi\) (provided such \(\pi\) exists). The implementation is based on MCMAS’s existing support for counterexample generation in CTL stemming from [13]. Since the LDLK model checking algorithm in MCMAS\(_{LDLK}\) builds a nondeterministic Büchi automaton for \(\neg \phi\) and looks for an accepting run, this run (if present) is also a valid counterexample. We can post-filter the states to those in which the path terminator is alive, to recover a finite trace counterexample.

This is particularly useful in practical contexts, as we report below.

### 5 EXPERIMENTAL RESULTS

**Experimental Setup.** To evaluate the proposed algorithms, we ran several experiments on virtual machines with two 2.70GHz CPUs and 16 GB of RAM, running Ubuntu v15.10 (Linux kernel v4.2). We evaluated the performance of MCMAS\(_{LDLK}\) in two ways:

1. For specifications that are expressible over both finite and infinite traces, we compared the performance of our proposed LDL\(_f\)K algorithm against that of the LDLK algorithms already implemented in MCMAS\(_{LDLK}\).

2. For specifications that do not have an analogue over infinite traces (e.g., properties involving termination), we simply evaluated the scalability of the approach. Observe that no other toolkit supports LDL\(_f\), nor LDL\(_f\)K, and a comparison against MCMAS\(_{LDLK}\) would not be meaningful in this case.

We modelled the popular Go-Back-N ARQ protocol [9] in ISPL, and used it as a test bed for evaluating our algorithms. Go-Back-N is a network communication protocol which achieves both delivery of messages in the presence of a faulty channel, and improved channel utilisation, especially in high-latency conditions.

Similarly to the Bit Transmission Protocol from [20], the system consists of two agents, a sender and a receiver, and the environment. The sender aims to communicate a vector of \(M\) packets to the receiver across a high-latency channel that may drop packets. Each of these packets is transmitted along with the appropriate sequence number in 1, \ldots, \(M\). The sender does not wait for an ack after each packet; it keeps transmitting up to a window of \(N\) unacknowledged packets. Upon receipt of a packet, the receiver sends an ack with the number of the last packet received in sequence. If the sender receives a duplicated ack or reaches the end of its transmission window, it restarts from the first unacknowledged packet.

In our model we use bits for data packets. This is sufficient to assert properties concerning protocol correctness. We also model a high-latency channel as one delivering messages with a delay of
There are a few exceptions to the general increasing trend noted above. One of these occurs when $M = 3$ and $N$ increases from 3 to 4; runtimes are almost identical. However, this is to be expected as the reachable state space is the same in both cases, the sender can send all the bits without receiving an ack. Indeed, the memory required for building the BDDs here was identical. We also observed occasional decreases in runtime or memory usage. We suspect that these arise from empirical efficiencies of BDDs, where the CUDD package was internally able to work better on a specific model. This is in line with other model checking experiments with BDDs [11, 31] and, in any case, these exceptions are small and infrequent.

In general, we observe that MCMAS$_{LDL_K}$ adds an overhead in both preprocessing and verification. This is also expected; preprocessing involves additional computation to derive $IS'$ and to translate the specifications; verification itself will be slower as both the size of the model and the specification have increased. However, the overall runtimes and memory usage are generally on the same order of magnitude, suggesting that the overhead is limited.

**Scalability Assessment.** For properties which do not have an analogue over infinite traces (for example, properties requiring termination), a comparison against MCMAS$_{LDL_K}$ would not be meaningful. Given this we here similarly evaluate how the performance of MCMAS$_{LDL_K}$ scales as $M$ and $N$ increase. To conduct this analysis we attempted to verify a complex specification – that if we abort the protocol immediately once the receiver learns about bit $M$, then the sender must know the receiver knows bit $M - N$ (where $M > N$). This may be written as $\phi_2$, where:

$$\phi_2 = ((-\phi_{KM})^*)[\phi_{KM}] \perp \langle \top \rangle^*[\phi_{KP}] \perp,$$

$$\phi_{KM} = K_R(0^M) \vee K_R(1_M)$$

$$\phi_{KP} = S(K_R(0^{M-N}) \vee K_R(1_{M-N}))$$

where $0_k$ is true iff bit $k$ is 0 (and respectively for $1_k$). Intuitively, the specification should be satisfied. The sender does not send bit $M$ until it has received an ack for at least $M-N$ bits; the receiver does not ack $M-N$ or greater until it receives bit $M-N$. Observe that it is possible to assess $\phi_2$ by verifying a stronger property over infinite traces, i.e., that the receiver cannot know bit $M$ without the sender knowing the receiver knows the bit $M - N$, i.e., $\phi_3 = [\top^*](-\phi_{KM} \land \neg\phi_{KP})$. However, this does not capture circumstances where the protocol is aborted. Consider that $\phi_2$ could still be satisfied even when the receiver knows bit $M$ but the sender does not know bit $M - N$, if we can guarantee that paths will not end in this state. Note that $\phi_2$ as written is unsatisfiable over infinite traces.

When testing MCMAS$_{DLK}$ against $\phi_2$ we encountered timeout outs, i.e., the checker was unable to verify $\phi_2$ on the model within 3 hours, even for smaller models. Observe that LDLK model checking is exponential in formula size and $\phi_2$ is of considerable length [30].

In an attempt to verify the protocol against the specification, we then proceeded to embed some of the formula assumptions directly in the model, at the cost of increasing its complexity. The rationale for this is that model checking LDLK is fixed parameter tractable [30], scaling polynomially in the model size. Specifically, we incorporated knowledge of when a trace was about to end into the path terminator. We implemented a different path terminator from that introduced in Section 4, shown in Figure 1. This path terminator explicitly tracks whether the current path is in its last
Agent AltPathTerminator

Vars:
  state : {alive, last, dead};
end Vars

Actions = {go, brake, stop};

Protocol:
  state = alive: {go, brake};
  state <> alive: {stop}; -- last or dead

end Protocol

Evolution:
  state = alive if (Action = go);
  state = last if (Action = brake);
  state = dead if (Action = stop);

end Evolution

Figure 1: Last-state-aware path terminator in ISPL. This example uses SingleAssignment semantics.

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>time (s) preprocessing</th>
<th>time (s) verification</th>
<th>BDD memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>18</td>
<td>1.040</td>
<td>0.342</td>
<td>40.15</td>
</tr>
<tr>
<td>4</td>
<td>18</td>
<td>1.062</td>
<td>0.698</td>
<td>52.62</td>
</tr>
<tr>
<td>5</td>
<td>18</td>
<td>5.667</td>
<td>1.235</td>
<td>56.27</td>
</tr>
<tr>
<td>6</td>
<td>18</td>
<td>6.028</td>
<td>2.726</td>
<td>59.35</td>
</tr>
<tr>
<td>8</td>
<td>18</td>
<td>28.565</td>
<td>8.658</td>
<td>61.33</td>
</tr>
<tr>
<td>10</td>
<td>18</td>
<td>19.674</td>
<td>6.753</td>
<td>58.76</td>
</tr>
<tr>
<td>12</td>
<td>18</td>
<td>33.191</td>
<td>52.081</td>
<td>81.54</td>
</tr>
<tr>
<td>15</td>
<td>18</td>
<td>164.842</td>
<td>66.038</td>
<td>82.03</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>235.340</td>
<td>82.401</td>
<td>107.45</td>
</tr>
<tr>
<td>4</td>
<td>18</td>
<td>2.244</td>
<td>1.496</td>
<td>57.62</td>
</tr>
<tr>
<td>5</td>
<td>18</td>
<td>7.286</td>
<td>2.681</td>
<td>56.42</td>
</tr>
<tr>
<td>6</td>
<td>18</td>
<td>15.986</td>
<td>6.265</td>
<td>65.61</td>
</tr>
<tr>
<td>8</td>
<td>18</td>
<td>32.395</td>
<td>43.358</td>
<td>73.13</td>
</tr>
<tr>
<td>10</td>
<td>18</td>
<td>87.725</td>
<td>68.088</td>
<td>86.23</td>
</tr>
<tr>
<td>12</td>
<td>18</td>
<td>94.001</td>
<td>95.753</td>
<td>115.05</td>
</tr>
<tr>
<td>15</td>
<td>18</td>
<td>516.412</td>
<td>201.195</td>
<td>207.00</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>1289.344</td>
<td>258.506</td>
<td>258.46</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>18.138</td>
<td>4.745</td>
<td>57.59</td>
</tr>
<tr>
<td>6</td>
<td>18</td>
<td>32.036</td>
<td>33.926</td>
<td>72.24</td>
</tr>
<tr>
<td>8</td>
<td>18</td>
<td>72.549</td>
<td>49.842</td>
<td>109.60</td>
</tr>
<tr>
<td>10</td>
<td>18</td>
<td>140.808</td>
<td>153.936</td>
<td>149.20</td>
</tr>
<tr>
<td>12</td>
<td>18</td>
<td>572.372</td>
<td>257.079</td>
<td>257.01</td>
</tr>
<tr>
<td>15</td>
<td>18</td>
<td>1623.293</td>
<td>656.292</td>
<td>486.81</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>3616.632</td>
<td>773.141</td>
<td>581.62</td>
</tr>
</tbody>
</table>

Table 2: Runtimes obtained by MCMAS_{LDLK} when verifying \( \phi_2 \) with an alternate path terminator over finite traces of the Go-Back-N ARQ protocol. Notice that \( \phi_2 \) requires \( M > N \).

state. We also modified the initial states to include those where the agent has state alive or last, as one-state paths are valid paths.

When converting the problem to infinite trace LTLK verification, this allowed us to verify the revised model against the specification:

\[
\phi_2' = (\langle \neg \phi_{KM} \rangle^\omega \phi_{KM} \land \text{Last}) \rightarrow (\langle T^\omega \rangle \phi_{KP} \land \text{Last})
\]

where Last is an atomic proposition holding in the last state of the path. Note that the presence of Last implies the path is still alive, and since we use MCMAS’s fairness constraints, we already only consider finite paths.

The set of reachable states for the amended model is approximately 3 times the size of the original state space (as opposed as “only” twice the size in the previous encoding). The experiments obtained are reported in Table 2. As expected, runtimes generally increase with M and N. However, this increase appears manageable; we were able to verify properties over state spaces of considerable size (e.g., for \( N = 4, M = 18 \), the reachable state space was of the order of \( 10^{13} \) states). Given the benchmarks, we deduce that the toolkit could conceivably verify even larger models.

Comparing the results in Table 2 with those obtained for \( \phi_1 \) in Table 1, we observe that, while preprocessing still dominates verification, this appears to be not as much as in the case of \( \phi_1 \). This is because \( \phi_2' \) contains more complex temporal semantics; so propositional shortcircuiting is less powerful. Interestingly, preprocessing times for \( \phi_2' \) are inconsistent with the finite trace preprocessing times for \( \phi_1 \), but they are generally slightly faster. The implementation of the path terminator is different, and in the case of \( \phi_2' \), it is optimised for the specification in question. We also note that memory usage is consistently higher, which is consistent with the aforementioned additional increase in the state space.

In addition to our experimental evaluation, we also checked the correctness of the implementation on a number of examples and testcases. Related to the case here discussed, we also verified that the specification \( \phi_2' \) differing from \( \phi_2 \) by replacing \( \phi_{KM} \) with \( K_5(K_0(0_{M-N+1}) \lor K_{1}(1_{M-N+1})) \) is indeed false on the protocol; the protocol allows the sender to send bit \( M \) without receiving an ack for bit \( M-N+1 \), and the channel might drop all of the receiver’s acks after \( M-N \). We verified \( \phi_2' \) using an analogous optimisation, and obtained similar runtimes. Indeed, the counterexample we obtained helped us understand why \( \phi_2' \) is false on the model.

6 CONCLUSIONS AND FURTHER WORK

In this paper we have contributed to the development of formalisms for reasoning about MAS under the assumption of finite traces. As we discussed, this is a recent development which has particular applications in AI. Specifically, we have introduced the logic LTLK, a temporal-epistemic logic in which the expressive temporal expressive given by LTLK can be used on finite paths. As discussed, a key feature of the logic is that the length of paths under consideration is not known to the agents before or during the execution, leading to a rich interplay between temporal and epistemic modalities.

We have also studied the model checking problem for LTLK and identified a reduction to the corresponding problem for LTLK (for a different model and for a different specification). We have proved that the reduction does not increase the overall complexity of the problem, which we showed to be PSPACE-complete. We showed the reduction is fixed parameter tractable; it is exponential in the size of specification formulae but not in the size of the model.

We have implemented and released an implementation of the algorithms presented as an extension of MCMAS_{LDLK}, an existing open-source toolkit for the verification of MAS. The results obtained point to slightly increased verification times and memory over the existing benchmarks for LTLK due to the transformation of the model and formula. These, however, appear limited and do not hinder the feasibility of the approach. The toolkit we put forward also supports counter-example generation, which is useful in practically interpreting model checking results.

In future work we intend to develop the methods here presented even further for planning and to seek application in the context of services by following the direction suggested in [39].