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 LDL_fK, 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 LDL_fK specifications and give algorithms for the reduction of LDL_fK 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

ACM Reference Format:
Jeremy Kong and Alessio Lomuscio. 2018. Model Checking Multi-Agent Systems against LDLK Specifications on Finite Traces. In *Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), Stockholm, Sweden, July 10–15, 2018,* IFAAMAS, 9 pages.

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 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 LDL_fK, 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 LDL_fK 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 nondeterministic 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.

Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), M. Dastani, G. Sukthankar, E. André, S. Koenig (eds.), July 10–15, 2018, Stockholm, Sweden.

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 $\mathrm{LDL}_f\mathrm{K}$ 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 stems from specification formulae, as opposed to the environment.

The rest of the paper is organised as follows. In Section 2, after summarising the semantics of interpreted systems, we define the syntax and the semantics of the logic $\mathrm{LDL}_f\mathrm{K}$ and define the corresponding model checking problem. In Section 3 we present novel algorithms for solving the model checking problem $\mathrm{LDL}_f\mathrm{K}$ by means of a reduction to the model checking problem (on a different model and for a different specification) for LDLK. We prove the algorithms are correct, and that they are optimal by showing the problem is PSPACE-complete. We continue in Section 4, where we report on a toolkit for the model checking of $\mathrm{LDL}_f\mathrm{K}$ built by extending the MCMAS experimental toolkit for LDLK. In Section 5 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 $IS = \langle \{L_i, Act_i, P_i, \tau_i\}_{i \in A \cup \{E\}}, I, h \rangle$, 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.
- $P_i : L_i \to 2^{Act_i} \backslash \{\emptyset\}$ is a *local protocol* for agent i, specifying which actions agent i can execute from each local state.
- $\tau_i : L_i \times Act_1 \times \ldots \times Act_n \times Act_E \to L_i$ 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 are consistent with their protocols; i.e., if $l'_i = \tau_i(l_i, a_1, \ldots, a_n, a_E)$ 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.*
- $h : AP \to 2^{L_1 \times \ldots \times L_n \times L_E}$, 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 τ_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 define the notion of a *path* on IS as an infinite sequence of global states $g_0, g_1, \ldots$ such that for each i, $g_i \in G$, and if $i > 0$, then there exist actions $a_1, \ldots, a_n, a_E$ so that for every local state, l'_j in g_i, $l'_j = \tau_j(l_j, a_1, \ldots, a_n, a_E)$ for each possible agent $j \in A \cup \{e\}$ (where l_j is the local state of agent j in state g_{i-1}).

We also allow referencing individual states on a path. We denote $\pi(k)$ to be the *k-th state of a path* $\pi = g_0, g_1, \ldots$, and π^k to be the *k-th suffix* of π, i.e. $\pi^k = g_k, g_{k+1}, \ldots$.

Finally, for each agent $i \in A$ we also define a projection $l_i : G \to L_i$. 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 $\mathrm{LDL}_f\mathrm{K}$, is identical to that of regular LDLK, as introduced in [30].

Definition 2. An *$\mathrm{LDL}_f\mathrm{K}$ formula* ϕ is constructed as follows:
$\langle \phi \rangle ::= p \mid \top \mid \neg\phi \mid \phi \wedge \phi \mid K_i\phi \mid E_\Gamma\phi \mid D_\Gamma\phi \mid C_\Gamma\phi \mid \langle\rho\rangle\,\phi$
$\langle \rho \rangle ::= \psi \mid \phi? \mid \rho + \rho \mid \rho; \rho \mid \rho^*$
p is a propositional atom; $i \in A \cup \{E\}$ is an agent, $\Gamma \subseteq A \cup \{E\}$ a set of agents, and ψ a formula not containing LDL operators ($\langle\rho\rangle\phi$) not in scope of an epistemic operator[1]. We allow standard propositional abbreviations, and a dual for the diamond operator: $[\rho]\phi = \neg\langle\rho\rangle\neg\phi$.

Intuitively, ρ 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 ρ, after which ϕ is true. The reading of $[\rho]\phi$ is that after every path prefix satisfying ρ, ϕ is true.

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

While the syntax of $\mathrm{LDL}_f\mathrm{K}$ 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.

- $\langle\top^*\rangle tun_1$. 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.
- $[\top^*](K_c tun_1 \to \langle\top;\top;(\top;\top)^*\rangle tun_2)$. 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.
- $[\top^*]\langle\top^*\rangle tun_1$. 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"[2].
- $[(\top;\top)^*]\langle(\top;\top)^*\rangle tun_1$. 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".
- $[(\neg K_2\langle\top^*\rangle tun_1?; tun_2)^*; K_2\langle\top^*\rangle tun_1?]\neg tun_2$. 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").
- $K_c\langle(tun_1; tun_2)^*\rangle[\top]\bot$. Note that this formula is unsatisfiable on infinite traces, since $[\top]\bot$ cannot be satisfied; it therefore is equivalent to falsity. However, on finite traces $[\top]\bot$ can hold if the trace has ended, since no prefix will match $\top$. Thus, on finite traces, the formula expresses that

[1] For example, $K_a\langle p\rangle q$ is allowed for ψ, but $\langle K_a p\rangle q$ is not.
[2] 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, $[\top]\bot$ allows us to characterise the end of a trace [17].

Given the above we conclude that LDL_fK allows us to render very expressive specifications over finite traces. We also observe that the meaning of LDL_fK formulae can be very different from that of syntactically equivalent LDLK formulae, suggesting that the expressivity of LDL_fK should be explored separately from LDLK.

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

Definition 3. Let $IS = \langle (L_i, Act_i, P_i, \tau_i)_{i \in A \cup \{E\}}, I, h\rangle$ 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: $IS, \pi, Last \models \phi$) is inductively defined as follows:

- $IS, \pi, Last \models p$ iff $\pi(0) \in h(p)$.
- $IS, \pi, Last \models \top$.
- $IS, \pi, Last \models \neg\phi$ iff it is not the case that $IS, \pi, Last \models \phi$.
- $IS, \pi, Last \models \phi_1 \wedge \phi_2$ iff it is the case that both $IS, \pi, Last \models \phi_1$ and $IS, \pi, Last \models \phi_2$.
- $IS, \pi, Last \models \langle\rho\rangle\phi$ iff there exists $0 \leq i \leq Last$ such that $(0, i) \in \mathcal{R}(\rho, \pi)$ and $IS, \pi^i, (Last - i) \models \phi$. Note that the latter term is well-defined, since $i \leq Last$.
- $IS, \pi, Last \models K_i\phi$ iff for every state $g \in G$ if $l_i(g) = l_i(\pi(0))$, then $IS, g \models \phi$.
- $IS, \pi, Last \models E_\Gamma\phi$ iff for every state $g \in G$ if $l_i(g) = l_i(\pi(0))$ for *some* agent $i \in \Gamma$ then $IS, g \models \phi$.
- $IS, \pi, Last \models D_\Gamma\phi$ iff for every state $g \in G$ if $l_i(g) = l_i(\pi(0))$ for *all* agents $i \in \Gamma$ then $IS, g \models \phi$.
- $IS, \pi, Last \models C_\Gamma\phi$ iff for every state $g \in G$ if $g_1, \ldots$ are states such that $l_i(g) = l_i(g_1)$, $l_{k_1}(g_1) = l_{k_1}(g_2), \ldots, l_j(g_n) = l_j(\pi(0))$ for some agents $i, j, k_1, \ldots \in \Gamma$ and positive integer n, then $IS, g \models \phi$.

The relation $\mathcal{R} \subseteq (\rho \times \pi) \times (\mathbb{N} \times \mathbb{N})$ is defined very similarly to LDLK, though we need to account for the finiteness of traces:

- $\mathcal{R}(\psi, \pi) = \{(i, i+1) : IS, \pi^i, Last - i \models \psi\}$
- $\mathcal{R}(\phi?, \pi) = \{(i, i) : IS, \pi^i, Last - i \models \phi\}$
- $\mathcal{R}(\rho + \rho', \pi) = \mathcal{R}(\rho, s) \cup \mathcal{R}(\rho', s)$
- $\mathcal{R}(\rho; \rho', \pi) = \{(i, j) : \exists k \text{ s.t. } (i, k) \in \mathcal{R}(\rho, \pi) \wedge (k, j) \in \mathcal{R}(\rho', \pi)\}$
- $\mathcal{R}(\rho^*, \pi) = \{(i, i)\} \cup \{(i, j) : \exists k \text{ s.t. } (i, k) \in \mathcal{R}(\rho, \pi) \wedge (k, j) \in \mathcal{R}(\rho^*, \pi)\}$

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 LDL_f specifications in this context as well. It is in the spirit of LTL_f and LDL_f 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 ρ are regular expressions over propositional or epistemic formulas ψ (with support for tests, which check that a given LDL_fK formula is true in a given state). These follow the standard PDL semantics regarding choice, composition and the Kleene star. $\langle\rho\rangle\phi$ is satisfied if and only if there exists some prefix of a path matching ρ, after which ϕ holds. Conversely, $[\rho]\phi$ means that after *every* prefix of a path matching ρ, ϕ must hold.

As in LDLK, for a global state $g \in G$, we say $IS, g \models \phi$ iff on every path π starting at g and every value of $Last$ we have $IS, \pi, Last \models \phi$.

Finally, we introduce the LDL_fK model checking problem: given an interpreted system IS, state g and LDL_fK formula ϕ, determine whether $IS, g \models \phi$.

3 MODEL CHECKING LDL_fK

In this section we introduce a model checking problem for LDL_fK and give tight bounds for the complexity of the problem.

We solve the LDL_fK 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 LDL_fK formulae to suitable LDLK formulae respecting when paths become inactive.

Formally, given an interpreted system IS, state g in IS and LDL_fK formula ϕ, we seek to find another interpreted system IS', state g' in IS' and LDLK formula ϕ', such that $IS, g \models_{LDL_f K} \phi$ iff $IS', g' \models_{LDLK} \phi'$. Note that the expression on the left uses LDL_fK 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 $IS = \langle (L_i, Act_i, P_i, t_i)_{i \in A \cup \{E\}}, I, h\rangle$, we define $IS' = \langle (L_i, Act_i, P_i, t_i)_{i \in A \cup \{E,P\}}, I', h'\rangle$, where P is as in Definition 4, I' 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 *Alive* proposition and where we assume that *Alive* 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 $IS = \langle (L_i, Act_i, P_i, t_i)_{i \in A \cup \{E\}}, I, h\rangle$, IS' as defined in Definition 5, π a path in IS and *Last* a nonnegative integer, the *corresponding path* π' 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, $l_P(\pi'(n)) = \begin{cases} Alive & n \leq Last \\ \neg Alive & n > Last \end{cases}$

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 π.

Determining ϕ'. We now present the details leading to the definition of ϕ' 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 ρ; 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 *Alive* predicate defined above.

Algorithm 1 Finite Restriction

INPUT: LDL$_f$K formula ϕ
OUTPUT: LDLK formula ϕ'
function FINITE-RESTRICT(ϕ)
  return $\langle \top^* \rangle(\neg Alive) \rightarrow$ LIVE-RESTRICT(ϕ)
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 ρ has been satisfied.

We also assume the existence of a function TRANSLATE-ALPHABET, which applies LIVE-RESTRICT on the formulae in any occurrences of ψ or ϕ?. This can be easily implemented, e.g., via recursive descent.

Notice that the separation of FINITE-RESTRICT and LIVE-RESTRICT is not strictly necessary for correctness (one could apply FINITE-RESTRICT throughout). However, we expect that using only 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 ϕ
OUTPUT: LDLK formula ϕ'
function LIVE-RESTRICT(ϕ)
  if ϕ is an atomic proposition p **then return** p
  else if $\phi = \top$ **then return** $\top$
  else if $\phi = \neg\phi_1$ **then return** $\neg$(LIVE-RESTRICT(ϕ_1))
  else if $\phi = \phi_1 \wedge \phi_2$ **then**
    return LIVE-RESTRICT(ϕ_1) $\wedge$ LIVE-RESTRICT(ϕ_2)
  else if $\phi = K_a\phi_1$ **then return** K_a(FINITE-RESTRICT(ϕ_1))
  else if $\phi = E_\Gamma\phi_1$ **then return** E_Γ(FINITE-RESTRICT(ϕ_1))
  else if $\phi = D_\Gamma\phi_1$ **then return** D_Γ(FINITE-RESTRICT(ϕ_1))
  else if $\phi = C_\Gamma\phi_1$ **then return** C_Γ(FINITE-RESTRICT(ϕ_1))
  else ρ' = TRANSLATE-ALPHABET(ρ) ▷ $\phi = \langle \rho \rangle \phi_1$
    return $\langle \rho' \rangle(Alive \wedge$ LIVE-RESTRICT(ϕ_1))
  end if
end function

LEMMA 3.1. *Let IS be an interpreted system, π a path in IS, Last a nonnegative integer and ϕ an LDL$_f$K formula. Let IS' be IS defined as in Definition 5, and π' defined as in Definition 6. Then,*

$$IS', \pi' \models \textit{FINITE-RESTRICT}(\phi) \text{ iff } IS', \pi' \models \textit{LIVE-RESTRICT}(\phi)$$

PROOF. By definition of π', P satisfies $\neg Alive$ after $Last+1$ states. Thus, $IS', \pi' \models \langle \top^* \rangle(\neg Alive)$. Now, consider that

$$\begin{aligned} & IS', \pi' \models \text{FINITE-RESTRICT}(\phi) \\ \Leftrightarrow\ & IS', \pi' \models \langle \top^* \rangle(\neg Alive) \rightarrow \text{LIVE-RESTRICT}(\phi) \\ \Leftrightarrow\ & IS', \pi' \models \text{LIVE-RESTRICT}(\phi) \end{aligned}$$ □

LEMMA 3.2. *Let IS be an interpreted system and ϕ an LDL$_f$K formula. Let IS' be IS defined as in Definition 5. Let π' be a path in IS' and suppose π' is not a corresponding path of any path in IS. Then, $IS', \pi' \models$ FINITE-RESTRICT(ϕ).*

PROOF. We first show that π' must have *Alive* permanently true. Suppose this is not true, for a contradiction. There then must be a first state where *Alive* is false, with index $n \geq 1$; we can choose $Last = n-1$. We can recover π by taking a projection of π', ignoring the path terminator's state. Then π' is the corresponding path for π and $Last = n - 1$. Then, by LDLK semantics, IS', π' does not satisfy $\langle \top^* \rangle(\neg Alive)$, so $IS', \pi' \models$ FINITE-RESTRICT(ϕ). □

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

PROOF. Observe that every corresponding path $\pi^{*\prime} \in P^c_{Ka}$ has $l_a(\pi^{*\prime}(0)) = l_a(\pi'(0))$ and is thus in Q_{Ka}. Notice that these are equal to $l_a(\pi^*(0))$ and $l_a(\pi(0))$ by construction, and *those* are equal by definition. Hence $P^c_{Ka} \subseteq Q_{Ka}$. Conversely, suppose for a contradiction that there is some path $\pi^{\dagger *}$ in Q_{Ka} that is a corresponding path of some π^{**} not in P_{Ka}. This means that $l_a(\pi^{**}(0)) \neq l_a(\pi(0))$, a contradiction since $l_a(\pi^{\dagger *}(0)) = l_a(\pi'(0))$. Thus, $Q_{Ka} \subseteq P^c_{Ka}$. Thus, the sets are equal. □

Lemma 3.4. *Let IS be an interpreted system, π be a path in IS, Last be a nonnegative integer and ϕ be an $LDL_f K$ formula. Then,*

$$IS, \pi, Last \models_{LDL_f K} \phi \text{ iff } IS', \pi' \models_{LDLK} \phi'$$

where: IS' is defined above; $\phi' =$ Finite-Restrict(ϕ); and π' is defined as in Definition 6.

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

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

$$IS, \pi, Last \models_{LDL_f K} \phi \text{ iff } IS', \pi' \models_{LDLK} \text{Live-Restrict}(\phi)$$

The Boolean cases are trivial. For the epistemic modalities, we first consider $K_a\phi$. Consider that $IS, \pi, Last \models_{LDL_f K} K_a\phi$ iff for every $g \in G$ with $l_a(g) = l_a(\pi(0))$, $IS, g \models_{LDL_f K} \phi$. By definition of satisfaction on states, this holds iff for every path π^* in IS such that $l_a(\pi^*(0)) = l_a(\pi(0))$ and value of $Last$, $IS, \pi^*, Last \models_{LDL_f K} \phi$.

We claim our previous assertion holds iff for every path $\pi^\dagger$ in IS' with $l_a(\pi^\dagger(0)) = l_a(\pi'(0))$, we have $IS', \pi^\dagger \models$ Finite-Restrict(ϕ).

Using Lemma 3.2, the non-corresponding paths must satisfy Finite-Restrict(ϕ). 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 π^*. Going forward, for each π^* and value of $Last$, the corresponding path $\pi^{*\prime}$ satisfies Finite-Restrict(ϕ) by the inductive hypothesis. Going backward, if $IS, \pi^\dagger \models$ Finite-Restrict(ϕ) and corresponds to path π^* and integer $Last$, $IS, \pi^*, Last \models \phi$ by the inductive hypothesis.

Finally, by definition of satisfaction on states we have our assertion holding iff for every state g' in IS' with $l_a(g') = l_a(\pi'(0))$, $IS', g' \models$ Finite-Restrict(ϕ). This precisely matches the semantics of $IS', \pi' \models K_a$(Finite-Restrict(ϕ)). The proofs for E_Γ, D_Γ and C_Γ are similar. This completes the proof for the epistemic cases.

Finally, consider the $\langle\rho\rangle\phi$ case. We have that

$IS', \pi' \models_{LDLK}$ Live-Restrict($\langle\rho\rangle\phi$) iff

$IS', \pi' \models_{LDLK} \langle\rho'\rangle(Alive \wedge$ Live-Restrict(ϕ)) iff

$\exists 0 \leq i.(0, i) \in \mathcal{R}(\rho', \pi') \wedge IS, \pi'^i \models_{LDLK} (Alive \wedge$ Live-Restrict(ϕ)) iff

$\exists 0 \leq i.(0, i) \in \mathcal{R}(\rho', \pi') \wedge IS, \pi'^i \models_{LDLK} Alive$ and

$IS, \pi'^i \models_{LDLK}$ Live-Restrict(ϕ) iff

$\exists 0 \leq i \leq Last.(0, i) \in \mathcal{R}(\rho', \pi') \wedge IS, \pi'^i \models_{LDLK}$ Live-Restrict(ϕ) iff

$\exists 0 \leq i \leq Last.(0, i) \in \mathcal{R}(\rho', \pi') \wedge IS, \pi^i, Last - i \models_{LDL_f K} \phi$

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

Next, we show that $\mathcal{R}_{LDL_f K}(\rho, \pi) = \mathcal{R}_{LDLK}(\rho', \pi')$ for arbitrary ρ, where $\rho' =$ Translate-Alphabet(ρ). We proceed by structural induction on ρ. We first consider the base cases $\mathcal{R}_{LDL_f K}(\psi, \pi)$ and $\mathcal{R}_{LDL_f K}(\phi_t?, \pi)$. Notice that these are defined as $\{(i, i+1) : IS, \pi^i, Last - i \models \psi\}$ and $\{(i, i) : IS, \pi^i, Last - i \models \phi_t\}$ respectively. Since ψ and ϕ_t here are subformulae of ϕ, we apply the (main) inductive hypothesis. We can thus rewrite the sets as $\{(i, i+1) : IS', \pi'^i \models$ Finite-Restrict(ψ)$\}$ and $\{(i, i) : IS', \pi'^i \models$ Finite-Restrict(ϕ_t)$\}$ respectively. We then convert instances of Finite-Restrict to Translate-Alphabet by Lemma 3.1 and definition of Translate-Alphabet. We then have $\mathcal{R}_{LDLK}$(Translate-Alphabet(ψ), π') and $\mathcal{R}_{LDLK}$(Translate-Alphabet(ϕ_t)?, π'), by 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

$$\exists 0 \leq i \leq Last.(0, i) \in \mathcal{R}(\rho, \pi) \wedge IS, \pi^i, Last - i \models_{LDL_f K} \phi$$

and from Definition 3, we have $IS, \pi, Last \models \langle\rho\rangle\phi$. □

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, $IS, g \models_{LDL_f K} \phi$ if and only if $IS', g' \models_{LDLK} \phi'$, where $\phi' =$ Finite-Restrict(ϕ) and g' is g augmented with the path terminator in the state alive.*

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

Thus, we have $IS', g' \models_{LDLK} \phi'$ if and only if for every path π' where the path terminator is eventually dead, we have $IS', \pi' \models_{LDLK}$ Finite-Restrict(ϕ). 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 $IS, \pi, Last \models_{LDL_f K} \phi$. Since π' was arbitrary apart from the eventual death restriction, we have $IS', g' \models_{LDLK} \phi'$ iff for every path π and value of $Last$ we have $IS, \pi, Last \models_{LDL_f K} \phi$, i.e. if and only if $IS, g \models_{LDL_f K} \phi$. □

Computational Complexity. We now turn to explore the complexity of model checking interpreted systems against LDL_fK specifications. We assume, as usual, that interpreted systems are given explicitly. We first bound the size of IS' and ϕ':

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[3]. 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 ϕ be an $LDL_f K$ formula. Let $\phi' =$ Finite-Restrict(ϕ). Then, the size of ϕ', $|\phi'|$ is polynomially bounded in $|\phi|$.*

Proof. Consider that ϕ' is generated by Algorithm 1. Consider the formation tree of ϕ. 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 subformulae, 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. □

[3]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_fK specifications is PSPACE-complete.*

Proof. We first show hardness. Since an LDL_f formula is an LDL_fK formula, and LDL_f model checking is PSPACE-complete hence PSPACE-hard [17], LDL_fK model checking is PSPACE-hard.

We claim that our approach for solving the model checking problem for LDL_fK can be implemented in polynomial space.

Constructing IS' and ϕ' 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' where both have the path terminator alive or both have it dead, and then for each edge (u, v) in the original transition relation, adding an edge from u with the path terminator alive to v with the terminator dead. This algorithm is polynomial time, and thus polynomial space.

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_fK by transforming the LDL_fK 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_fK 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_{LDL_fK}, are available from [40].

MCMAS_{LDL_fK} 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_{LDL_fK} 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 ϕ to ϕ' 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(ϕ) can simply return Live-Restrict(ϕ) 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_{LDL_fK} also supports counterexample generation; this involves finding a finite trace π in IS where $IS, \pi \not\models \phi$ (provided such π 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_{LDL_fK} in two ways:

(1) For specifications that are expressible over both finite and infinite traces, we compared the performance of our proposed LDL_fK 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_fK, 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 unacked 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

		time (s)				BDD memory (MB)	
		preprocessing		verification			
N	M	infinite	finite	infinite	finite	infinite	finite
2	3	0.557	0.766	0.026	0.067	13.54	17.33
	4	1.058	2.288	0.052	0.208	20.12	33.00
	5	1.827	3.972	0.071	0.293	20.37	38.25
	6	2.711	4.741	0.131	0.261	29.93	35.38
	8	22.896	15.175	0.397	1.615	44.46	48.19
	10	19.586	27.044	0.563	1.954	48.92	49.22
	12	30.337	47.652	1.035	1.714	59.08	57.61
	15	109.510	268.017	1.520	4.175	56.32	65.10
3	3	0.784	1.086	0.048	0.113	18.84	19.74
	4	1.828	1.798	0.106	0.221	33.36	36.06
	5	5.113	8.525	0.200	0.851	42.63	58.56
	6	8.668	14.193	0.324	1.144	55.07	58.11
	8	28.923	65.390	1.029	5.509	58.15	68.40
	10	76.890	144.600	2.930	7.861	62.96	71.71
	12	106.773	118.334	3.969	10.135	74.82	86.30
	15	398.878	846.602	6.130	32.217	86.32	195.21
4	3	0.784	1.084	0.049	0.110	18.84	19.74
	4	2.913	3.812	0.184	0.304	36.35	40.85
	5	9.696	11.302	0.396	1.217	49.96	57.13
	6	19.171	43.746	0.561	6.002	57.23	62.40
	8	79.054	95.341	3.111	8.920	71.55	69.45
	10	101.781	271.890	5.096	22.547	85.59	99.28
	12	467.855	587.106	9.950	33.608	120.24	156.53
	15	1469.131	2094.340	25.387	70.775	214.56	307.58

Table 1: Runtimes for MCMAS$_{LDLK}$ and MCMAS$_{LDL_fK}$ when verifying ϕ_1 on infinite and finite traces of the Go-Back-N ARQ protocol.

three rounds, possibly dropping (but never corrupting) packets. We refer to the source of the model for more details [40].

Experimental comparison against MCMAS$_{LDLK}$. We compared the performance of MCMAS$_{LDLK}$ and MCMAS$_{LDL_fK}$ on a model of the Go-Back-N protocol against specifications expressed in both LDLK and LDL$_f$K. We verified on both infinite and finite traces that the sender and receiver are never in conflict on the bits the receiver knows. Intuitively, this is true as the environment does not corrupt the value of the bits. This may be specified as

$$\phi_1 = [\top^*]\left(\wedge_{i=1}^{M}(\neg mismatch_i)\right)$$

where $mismatch_i$ holds iff the receiver has received a bit for position i that is different from that of the sender. The results obtained are presented in Table 1.

Notice that unlike in standard verification experiments, we document the preprocessing time for infinite and finite traces separately. This is because we have a parsing load when building the new model, and, by adding a variable, we also double the size of the model; so reachability computations are likely to take longer.

We observe that generally as M and N increase, runtime for both preprocessing and verification increases. This holds for both verification over infinite and finite traces. This is unsurprising, as increases in M and N generally result in a larger models. As M increases, the number of bits to be conveyed increases, and as N increases the allowed "lag" between the sender and receiver increases, resulting in a larger state space. Note that although the size of the formula appears to increase linearly in M, the propositional shortcircuiting optimisation from [30] limits its overhead.

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$_{LDL_fK}$ 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$_{LDLK}$ would not be meaningful. Given this we here similarly evaluate how the performance of MCMAS$_{LDL_fK}$ 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 ϕ_2, where:

$$\begin{aligned}\phi_2 &= \langle(\neg\phi_{KM})^*\rangle[\phi_{KM}]\bot \rightarrow \langle\top^*\rangle[\phi_{KP}]\bot\\ \phi_{KM} &= K_R(0_M) \vee K_R(1_M)\\ \phi_{KP} &= K_S(K_R(0_{M-N}) \vee K_R(1_{M-N}))\end{aligned}$$

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 ϕ_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. $\phi_3 = [\top^*]\neg(\phi_{KM} \wedge \neg\phi_{KP})$. However, this does not capture circumstances where the protocol is aborted. Consider that ϕ_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 ϕ_2 as written is unsatisfiable over infinite traces.

When testing MCMAS$_{LDLK}$ against ϕ_2 we encountered timeouts, i.e., the checker was unable to verify ϕ_2 on the model within 3 hours, even for smaller models. Observe that LDLK model checking is exponential in formula size and ϕ_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
end Agent
```

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

		time (s)		BDD memory (MB)
N	M	preprocessing	verification	
2	3	1.040	0.342	40.15
	4	1.062	0.698	52.62
	5	5.667	1.235	56.27
	6	6.028	2.726	59.35
	8	28.563	8.658	61.33
	10	19.674	6.753	58.76
	12	33.191	52.081	81.54
	15	164.842	66.038	82.03
	18	235.340	82.401	107.45
3	4	2.244	1.496	57.62
	5	7.286	2.681	56.42
	6	15.986	6.265	65.61
	8	32.395	43.358	73.13
	10	87.725	68.088	86.23
	12	94.001	95.753	115.05
	15	516.412	201.195	207.00
	18	1289.344	258.506	238.46
4	5	10.138	4.745	57.59
	6	32.036	33.926	72.24
	8	72.549	49.842	109.60
	10	140.808	153.936	149.20
	12	572.372	257.079	257.01
	15	1623.293	656.292	486.81
	18	3616.632	773.141	581.62

Table 2: Runtimes obtained by MCMAS$_{LDL_fK}$ when verifying ϕ_2^* with an alternate path terminator over finite traces of the Go-Back-N ARQ protocol. Notice that ϕ_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 LDLK verification, this allowed us to verify the revised model against the specification:

$$\phi_2^* = \langle(\neg\phi_{KM})^*\rangle(\phi_{KM} \wedge Last) \rightarrow \langle\top^*\rangle(\phi_{KP} \wedge Last)$$

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 as 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 ϕ_1 in Table 1, we observe that, while preprocessing still dominates verification, this appears to be not as much as in the case of ϕ_1. This is because ϕ_2^* contains more complex temporal semantics; so propositional shortcircuiting is less powerful. Interestingly, preprocessing times for ϕ_2^* are inconsistent with the finite trace preprocessing times for ϕ_1, but they are generally slightly faster. The implementation of the path terminator is different, and in the case of ϕ_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 ϕ_2' differing from ϕ_2 by replacing ϕ_{KP} with $K_S(K_R(0_{M-N+1}) \vee K_R(1_{M-N+1}))$ 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 ϕ_2' using an analogous optimisation, and obtained similar runtimes. Indeed, the counterexample we obtained helped us understand why ϕ_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 LDL$_f$K, a temporal-epistemic logic in which the expressive temporal expression given by LDL 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 LDL$_f$K and identified a reduction to the corresponding problem for LDLK (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 LDLK 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].

REFERENCES

[1] T. Ågotnes, V. Goranko, W. Jamroga, and M. Wooldridge. 2015. Knowledge and Ability. In *Handbook of Logics for Knowledge and Belief*. College Publications.

[2] N. Alechina, B. Logan, H. N. Nguyen, F. Raimondi, and L. Mostarda. 2015. Symbolic Model-checking for Resource-Bounded ATL. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS15)*. 1809–1810.

[3] R. Alur, L. de Alfaro, R. Grosu, T. Henzinger, A. Thomas, M. Kang, C. Kirsch, R. Majumdar F. Mang, and B-Y. Wang. 2001. jMocha: A model checking tool that exploits design structure. In *Proceedings of the 23rd International Conference on Software Engineering (ICSE01)*. IEEE, 835–836.

[4] É. André, L. Petrucci, W. Jamroga, M. Knapik, and W. Penczek. 2017. Timed ATL: Forget Memory, Just Count. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems (AAMAS17)*. ACM, 1460–1462.

[5] A. Bauer and P. Haslum. 2011. LTL goal specifications revisited. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI10)*. IOS Press, 881–886.

[6] F. Belardinelli, A. Lomuscio, and J. Michaliszyn. 2016. Agent-based Refinement for Predicate Abstraction of Multi-Agent Systems. In *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI16)*. IOS Press, 286–294.

[7] N. Bulling and W. Jamroga. 2014. Comparing variants of strategic ability: how uncertainty and memory influence general properties of games. *Autonomous Agents and Multi-Agent Systems* 28, 3 (2014), 474–518.

[8] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. 1992. Symbolic Model Checking: 10^{20} States and Beyond. *Information and Computation* 98, 2 (1992), 142–170.

[9] H. O. Burton and D. D. Sullivan. 1972. Errors and error control. In *Proceedings of the IEEE*, Vol. 60. IEEE.

[10] A. Camacho, E. Triantafillou, C. Muise, J. A. Baier, and S. A. McIlraith. 2017. Non-Deterministic Planning with Temporally Extended Goals: LTL over finite and infinite traces. *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI17)*, 3716–3724.

[11] P. Cermák, A. Lomuscio, F. Mogavero, and A. Murano. 2014. MCMAS-SLK: A Model Checker for the Verification of Strategy Logic Specifications. In *Proceedings of the 26th International Conference on Computer Aided Verification (CAV14) (Lecture Notes in Computer Science)*, Vol. 8559. Springer, 525–532.

[12] P. Cermák, A. Lomuscio, F. Mogavero, and A. Murano. 2015. Verifying and Synthesising Multi-Agent Systems against One-Goal Strategy Logic Specifications. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI15)*. AAAI Press, 2038–2044.

[13] E. M. Clarke, S. Jha, Y. Lu, and H. Veith. 2002. Tree-like Counterexamples in Model Checking. In *Proceedings of the 17th Symposium on Logic in Computer Science (LICS02)*. 19–29.

[14] M. Cohen, M. Dam, A. Lomuscio, and H. Qu. 2009. A Symmetry Reduction Technique for Model Checking Temporal-Epistemic Logic. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI09)*. 721–726.

[15] G. De Giacomo, R. De Masellis, M. Grasso, F. M. Maggi, and M. Montali. 2014. Monitoring Business Metaconstraints Based on LTL and LDL for Finite Traces. In *Proceedings of the 12th International Conference on Business Process Management (BPM14) (Lecture Notes in Computer Science)*, Vol. 8659. Springer, 1–17.

[16] G. De Giacomo, R. De Masellis, and M. Montali. 2014. Reasoning on LTL on Finite Traces: Insensitivity to Infiniteness. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI14)*. 1027–1033.

[17] G. De Giacomo and M. Y. Vardi. 2013. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'13)*. 854–860.

[18] G. De Giacomo and M. Y. Vardi. 2015. Synthesis for LTL and LDL on Finite Traces. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI15)*. AAAI Press, 1558–1564.

[19] G. De Giacomo and M. Y. Vardi. 2016. $\mathrm{LTL_f}$ and $\mathrm{LDL_f}$ Synthesis under Partial Observability. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI16)*. IJCAI/AAAI Press, 1044–1050.

[20] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. 1995. *Reasoning about Knowledge*. MIT Press, Cambridge.

[21] P. Gammie and R. van der Meyden. 2004. MCK: Model Checking the Logic of Knowledge. In *Proceedings of 16th International Conference on Computer Aided Verification (CAV04) (Lecture Notes in Computer Science)*, Vol. 3114. Springer, 479–483.

[22] J. A. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud. 2000. Introducing OBJ. In *Software Engineering with OBJ*. Springer, 3–167.

[23] J. Gutierrez, G. Perelli, and M. Wooldridge. 2017. Iterated Games with LDL Goals over Finite Traces. In *Proceedings of the 16th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS17)*. IFAAMAS Press, 696–704.

[24] K. Havelund and G. Rosu. 2001. *Testing linear temporal logic formulae on finite execution traces.* Technical Report 01.08. RIACS.

[25] W. van der Hoek and M. Wooldridge. 2002. Tractable multiagent planning for epistemic goals. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS02)*. ACM Press, 1167–1174.

[26] W. Jamroga and A. Murano. 2014. On module checking and strategies. In *Proceedings of the 14th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS14)*. IFAAMAS, 701–708.

[27] M. Kacprzak, W. Nabialek, A. Niewiadomski, W. Penczek, A. Pólrola, M. Szreter, B. Woźna, and A. Zbrzezny. 2008. VerICS 2007 - a Model Checker for Knowledge and Real-Time. *Fundamenta Informaticae* 85, 1 (2008), 313–328.

[28] M. Kacprzak and W. Penczek. 2004. A SAT-based approach to unbounded model checking for alternating-time temporal epistemic logic. *Synthese* 142, 2 (2004), 203–227.

[29] M. Kacprzak and W. Penczek. 2005. Fully Symbolic Unbounded Model Checking for Alternating-time Temporal Logic. *Autonomous Agents and Multi-Agent Systems* 11, 1 (2005), 69–89.

[30] J. Kong and A. Lomuscio. 2017. Model Checking Multi-Agent Systems against LDLK Specifications. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI17)*. AAAI Press, 1138–1144.

[31] J. Kong and A. Lomuscio. 2017. Symbolic Model Checking Multi-Agent Systems against CTL*K Specifications. In *Proceedings of the 16th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS17)*. IFAAMAS Press, 114–122.

[32] O. Kupferman, M. Y. Vardi, and P. Wolper. 2001. Module checking. *Information and Computation* 164, 2 (2001), 322–344.

[33] M. Kwiatkowska, A. Lomuscio, and H. Qu. 2010. Parallel Model Checking for Temporal Epistemic Logic. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI10)*. IOS Press, 543–548.

[34] A. Lomuscio and J. Michaliszyn. 2015. Verifying Multi-Agent Systems by Model Checking Three-valued Abstractions. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS15)*. 189–198.

[35] A. Lomuscio and J. Michaliszyn. 2016. Verification of Multi-Agent Systems via Predicate Abstraction against ATLK specifications. In *Proc. of the 15th Int. Conference on Autonomous Agents and Multiagent Systems (AAMAS16)*. 662–670.

[36] A. Lomuscio, W. Penczek, and B. Woźna. 2007. Bounded model checking knowledge and real time. *Artificial Intelligence* 171, 16-17 (2007), 1011–1038.

[37] A. Lomuscio, H. Qu, and F. Raimondi. 2017. MCMAS: An Open-Source Model Checker for the Verification of Multi-Agent Systems. *Software Tools for Technology Transfer* 19, 1 (2017), 9–30.

[38] A. Lomuscio and F. Raimondi. 2006. Model checking knowledge, strategies, and games in multi-agent systems. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS06)*. ACM Press, 161–168.

[39] F. Maggi, M. Montali, M. Westergaard, and W. van der Aalst. 2011. Monitoring Business Constraints with Linear Temporal Logic: An Approach Based on Colored Automata. *Business Process Management* (2011), 132–147.

[40] $\mathrm{MCMAS_{LDL_fK}}$. 2017. http://vas.doc.ic.ac.uk/. (2017).

[41] W. Penczek and A. Lomuscio. 2003. Verifying Epistemic Properties of multi-agent systems via bounded model checking. *Fundamenta Informaticae* 55, 2 (2003), 167–185.

[42] J. Torres and J. A. Baier. 2015. Polynomial Time Reformulations of LTL Temporally Extended Goals into Final-State Goals. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI15)*. AAAI Press, 1696–1703.

[43] M. Y. Vardi. 2011. The rise and fall of linear time logic. In *2nd International Symposium on Games, Automata, Logics, and Formal Verification.*