Model Checking Multi-Agent Systems against LDLK Specifications

Jeremy Kong and **Alessio Lomuscio**
Department of Computing, Imperial College London
jeremykong91@gmail.com, a.lomuscio@imperial.ac.uk

Abstract

We define the logic LDLK, a formalism for specifying multi-agent systems. LDLK extends LDL with epistemic modalities, including common knowledge, for reasoning about evolution of knowledge states of agents in the system. We show that the problem of verifying multi-agent systems against LDLK specifications is PSPACE-complete. We give an algorithm for practical verification of multi-agent systems specified in LDLK. We show that the model checking algorithm, based on alternating automata and ϵ-NFAs, is fixed parameter tractable, scalable in the size of models analysed, and amenable to symbolic implementation on OBDDs. We introduce MCMAS$_{LDLK}$, an extension of the open-source model checker MCMAS implementing the algorithm presented, and discuss the experimental results obtained.

1 Introduction

Linear Dynamic Logic (LDL) has recently been put forward as an attractive logic for reasoning about processes [Vardi, 2011]. LDL extends Linear Temporal Logic [Pnueli, 1977] by replacing the linear temporal operators with regular expressions in a way that is similar to that of Propositional Dynamic Logic [Fischer and Ladner, 1977]. From the theoretical point of view, LDL is attractive because it can express all ω-regular properties and is therefore equivalent to monadic second-order logic [Vardi, 2011]. From an application point of view the expressiveness of LDL has been positively evaluated in a number of contexts, including monitoring business processes [De Giacomo *et al.*, 2014]. A major feature of interest in LDL is the fact that its model checking problem is in PSPACE [Faymonville and Zimmermann, 2015]. So while LDL is considerably more expressive than LTL, their model checking problem has the same complexity.

While much progress has been made recently on LDL, including an analysis of the synthesis problem on infinite and finite traces [De Giacomo and Vardi, 2015; 2016], from an agent perspective we highlight two apparent limitations.

Firstly, from a theoretical standpoint, while LDL can express sophisticated temporal or action properties, no proposal has so far been made to augment it with modalities to represent the agents' mental states. This has been conducted successfully for a range of temporal logics in the context of epistemic logic and has resulted in the specification languages LTLK [Fagin *et al.*, 1995] and CTLK [Penczek and Lomuscio, 2003] which have attracted interests both in theory and applications. Due to the expressiveness of LDL the same exercise could be of even higher significance in this context.

Secondly, from an application point of view, no practical algorithm nor toolkit exists for verifying systems against LDL specifications. Indeed, while the computational complexity of the logic has been explored, no *symbolic* model checking algorithm has so far been put forward. Yet symbolic toolkits [Gammie and van der Meyden, 2004; Kacprzak *et al.*, 2008; Lomuscio *et al.*, 2015] are essential for verifying prototype designs in multi-agent systems.

In this paper we attempt to make a contribution addressing these two aspects. Firstly, in Section 2, we define LDLK, an extension of LDL with knowledge operators, including common knowledge, and its semantics. In Section 3 we put forward a model checking algorithm for verifying multi-agent systems against LDLK specifications. A key feature of the approach is that the procedure is based on alternating automata and amenable to symbolic implementation. In Section 4 we present MCMAS$_{LDLK}$, a model checker for verification of multi-agent systems against LDLK specifications. MCMAS$_{LDLK}$ extends the open-source model checker MCMAS, by implementing the algorithm of Section 3 for LDLK. We show the attractiveness of the experimental results obtained and conclude in Section 5.

Related Work. The verification algorithm we introduce for model checking multi-agent systems against LDLK is inspired by the one put forward in [Faymonville and Zimmermann, 2015], which showed that determining satisfiability of parametric LDL specifications is in PSPACE. However, their construction is focused on showing decidability as opposed to practical implementation; specifically, the algorithm does not explicitly address how to handle a potentially unbounded number of ϵ-paths when trying to quantify over all prefixes of a path. The construction we introduce here includes a mechanism for limiting the paths being considered. Our approach is not polynomial space as we eagerly build the automata; but, as we show, still leads to an efficient symbolic implementation in cases of practical interest we tested. Additionally, [Faymonville and Zimmermann, 2015] does not support epistemic specifications and no implementation is put forward.

Our implementation is based on the open-source model checker MCMAS [Lomuscio *et al.*, 2015]. However, MCMAS currently only supports CTLK; LDLK significantly extends the range of specifications supported. Note also that CTLK is implemented in MCMAS by a traditional labelling algorithm. Instead, we here use alternating automata, which are then converted to symbolic Büchi automata by using the symbolic breakpoint construction from [Bloem *et al.*, 2006]. So, while MCMAS$_{LDLK}$ shares with MCMAS the libraries for parsing the input and constructing some intermediate structures, the verification algorithm is entirely different.

MCMAS has also been extended to address the problem of verifying strategic ability in multi-agent systems; support has been developed for ATL and SL, extended with suitable epistemic operators [Cermák *et al.*, 2014]. However, our work is focused on improving temporal expressivity (as opposed to additional modalities).

The model checker NuSMV [Cimatti *et al.*, 1999] supports a fragment of Property Specification Language (PSL) which, like LDL, can express all ω-regular properties [Cimatti *et al.*, 2006]. However, this is limited to properties also expressible in LTL or CTL. We are not aware of any open-source model checker supporting full PSL.

The model checker MCK [Gammie and van der Meyden, 2004] supports the verification of CTL*K properties. Differently from MCMAS which is limited to observational semantics, MCK also supports clock semantics and perfect recall (for one agent only). However, CTL*K and LDLK have incomparable expressivity (the ability of CTL*K to reason temporally about a specific path is similar to that of LTL, which cannot express parity [Wolper, 1981]); so the functionality of MCK is different from that of MCMAS$_{LDLK}$.

2 Linear Dynamic Logic with Knowledge

Interpreted systems, introduced in [Fagin *et al.*, 1995], provide a formalism for reasoning about temporal and epistemic properties of MAS. In the approach given in [Lomuscio *et al.*, 2015], agents have locally defined transition relations, and the global transition relation of the system results from the composition of these. We take $A = \{1, \ldots, n\}$ as a set of agents, and e as a special agent called the environment.

Definition 1 (Interpreted Systems)**.** An *interpreted system* is 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 the *global transition relation* T to be the composition of the τ_i, and the set of *global reachable states* $G \subseteq L_1 \times \ldots \times L_n \times L_e$ to be the set of states reachable through zero or more applications of T on I. We also define an evolution *path* for IS, as an infinite sequence of global states $g_0, g_1, \ldots$ such that for every i, $g_i \in G$, and if $i > 0$ then there exists actions $a_1, \ldots, a_n, a_e$ such 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}). Furthermore, we denote with $\pi(k)$ the *k-th state* of a path $\pi = g_0, g_1, \ldots$, and with π^k the *k-th suffix* of a path π; i.e., $\pi^k = g_k, g_{k+1}, \ldots$. We also define, for each agent $i \in A$ a projection operator $l_i : G \to L_i$ which returns the local state of agent i in a given global state.

We now introduce Linear Dynamic Logic with Knowledge (LDLK) by augmenting LDL with epistemic operators.

Definition 2 (LDLK Syntax)**.** An *LDLK formula* ϕ is constructed by the following grammar:

$$\langle\phi\rangle ::= p \mid \top \mid \neg\phi \mid \phi \wedge \phi \mid K_i\phi \mid E_\Gamma\phi \mid D_\Gamma\phi \mid C_\Gamma\phi \mid \langle\rho\rangle\,\phi$$
$$\langle\rho\rangle ::= \psi \mid \phi? \mid \rho + \rho \mid \rho;\rho \mid \rho^*$$

where p is a propositional atom, $i \in A \cup \{e\}$ an agent, $\Gamma \subseteq A \cup \{e\}$ a set of agents and ψ a formula free of dynamic modalities ($\langle\rho\rangle\phi$). We allow the standard abbreviations from propositional logic, and define $[\rho]\phi = \neg\langle\rho\rangle\neg\phi$.

LDLK enables us to express LDL specifications in an epistemic setting; these are not expressible in LTLK, CTLK, nor CTL*K. In the context of the train gate controller scenario [van der Hoek and Wooldridge, 2002], these include:

- *Specifications involving parity*, as in "At even states in the run, train 1 is in the tunnel.": $[(\top;\top)^*]tun_1$;
- *Specifications involving alternating sequences*, as in "It is common knowledge between the trains that they will alternately occupy the tunnel until one breaks down.": $C_{\{1,2\}}[(tun_1; tun_2)^*](break_1 \vee break_2)$;
- *Looping constructions*, as in "Until train 2 knows train 1 will eventually enter the tunnel, train 2 stays outside the tunnel; once it knows this, train 2 enters the tunnel.": $[(\neg K_2\langle\top^*\rangle tun_1?; \neg tun_2)^*; K_2\langle\top^*\rangle tun_1?]tun_2$;
- *Complex regular properties*, as in "When train 1 enters the tunnel, it leaves within one step; the controller then knows that the tunnel is intermittently occupied until train 1 re-enters.": $[\top^*](tun_1 \to \langle(\top + \top?); \neg tun_1\rangle(K_C[(tun_1 \vee tun_2; \top)^*]tun_1))$;
- *Complex combinations of epistemic operators and regular expressions*, as in "The controller knows that after the tunnel is empty for two rounds, the trains will both know that the controller breaks in an odd number of rounds.": $K_C[\top^*; (empty; empty)]E_{\{1,2\}}\langle\top; (\top;\top)^*\rangle break_C$.

None of the properties above is expressible in CTL*K. As previously mentioned, LDL has been employed in a variety of applications including business processes [De Giacomo *et al.*, 2014]. By combining LDL with epistemic logic we obtain a very expressive language for expressing MAS specifications.

We now define the satisfaction relation.

Definition 3 (Satisfaction)**.** Consider the interpreted system $IS = \langle (L_i, Act_i, P_i, \tau_i)_{i \in A \cup \{e\}}, I, h \rangle$ and let π be a path.

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

The relation $\mathcal{R} \subseteq (\rho \times \pi) \times (\mathbb{N} \times \mathbb{N})$ is inductively defined:

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

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

Note that, as in mainstream approaches, we here adopt observational semantics for epistemic modalities; intuitively, agents' knowledge is defined on their local state only.

In LDLK the path expressions ρ in the dynamic modalities behave similarly to constructs from PDL:

- ψ checks if the formula ψ (by definition not containing dynamic modalities, but possibly containing epistmic ones) holds at the current state.
- $\phi?$ checks if the LDLK formula ϕ holds at the current state. Unlike ψ, though, the evaluation of $\phi?$ does not "progress" along the path – that is, in a composition like $\phi?; \rho$ we are checking that ϕ is true in this state, and that the path from this *same* state satisfies ρ.
- $\rho + \rho'$ models the *nondeterministic selection* between two path expressions; it checks whether *either of* the path expressions ρ or ρ' is satisfied by this path.
- $\rho; \rho'$ models the *sequential composition* of two path expressions; it checks whether it is possible for the current path to satisfy ρ *and then* ρ'.
- $\langle\rho^*\rangle$ models *zero or more instances of* ρ, similarly to Kleene star for regular expressions.

Given the above, the intuitive meaning of $\langle\rho\rangle\phi$ is that there exists some prefix matching ρ, after which ϕ holds; the dual $[\rho]\phi$ holds iff after *every* prefix matching ρ, ϕ holds.

Given an interpreted system IS, a state g and an LDLK formula ϕ, the model checking problem involves determining whether $IS, g \models \phi$.

3 Symbolic Model Checking LDLK

We now introduce an approach for verifying an interpreted systems IS against an LDLK specification ϕ. This is reported in Algorithm 1. As we show below, we have that $IS \models \phi$ iff I and the set of states returned by Algorithm 1 on $\neg\phi$ are disjoint. Since steps 3 and 4 follow established techniques, we here report steps 1 and 2 only.

Algorithm 1 Symbolic LDLK Model Checking Algorithm

```
1. Recursively descend $\phi$ and construct an LDL formula $\phi'$
   that holds in the same states as $\phi$ in $IS$.
2. Construct an alternating automaton for $\phi'$, encapsulating
   an accepting run.
3. Convert the alternating automaton to a symbolic Büchi
   automaton via the constructions in [Bloem et al., 2006].
4. Build the product of the Büchi automaton and $IS$; find
   states admitting an accepting run [Burch et al., 1990].
```

Translating LDLK to LDL. Given an LDLK formula, we first determine the states in which any epistemic subformulae hold, and introduce fresh atomic propositions holding in precisely the same states. This involves recursively computing the states in which the formula for which we are evaluating knowledge holds, and then using the bottom-up labelling algorithm presented in [Raimondi and Lomuscio, 2004]. The construction always terminates, since formulae are of finite size and thus have a finite nesting depth.

For example, to translate the LDLK formula $\phi = \langle (K_a(r)? + K_b(q)?); (K_b(q)? + K_b(p)?); \top \rangle K_a(r)$ we first determine the states in which $K_a(r)$, $K_b(q)$ and $K_b(p)$ each hold. We then define α, β and γ to hold in the respective sets of states, and obtain the formula $\phi' = \langle (\alpha? + \beta?); (\beta? + \gamma?); \top \rangle \alpha$, which in our model holds in the same states as ϕ.

Constructing the Alternating Automaton. Our approach builds upon the construction given in [Faymonville and Zimmermann, 2015] for constructing an alternating automaton that accepts the path π if and only if $IS, \pi \models \phi$. Given an LDL formula ϕ, resulting from the translation above, we first convert it into negation normal form (NNF), so that all negations appear before atomic propositions.

The translation of ϕ (in NNF) to an alternating automaton then largely follows that in [Faymonville and Zimmermann, 2015]. The propositional cases are straightforward. For each dynamic modality $\langle\rho\rangle\psi$ or $[\rho]\psi$ in ϕ, we use an extension of Thompson's construction [Thompson, 1968] to build a nondeterministic finite automaton with silent ϵ moves (ϵ-NFA).

We adopt their definition of *marked ϵ-NFA*.

Definition 4 (Marked ϵ-NFA)**.** A marked ϵ-NFA is a tuple $A = (\Sigma, S, s^0, R, f, m)$ where Σ is an alphabet, S is a finite nonempty set of *states*, s^0 is an *initial state*, f is an *accepting state*, $R : (S \times (\Sigma \cup \{\epsilon\})) \to 2^S$ is a (partial) *transition function* and $m : S \to \phi$ is a (partial) *marking function* assigning LDL formulae to states. Further to the above, if $\pi = s_0 \ldots s_n$ is a path in a marked ϵ-NFA, we define the markings over π, $m(\pi) = \{m(s_i) \mid i \in \{0, \ldots, n\}\}$. In our usage, we build marked ϵ-NFAs for each dynamic modality $\langle\rho\rangle\psi$ or $[\rho]\psi$ in the formula we are checking; the marking function is concerned with tests appearing in the relevant ρ. The overall

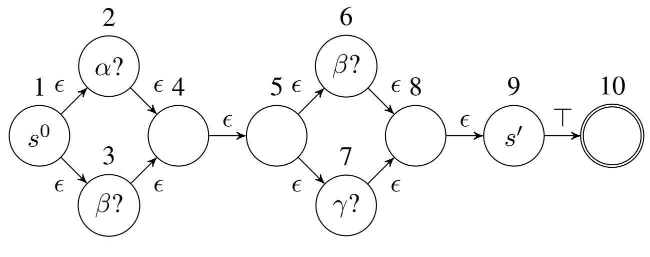

Figure 1: Marked ϵ-NFA for $(\alpha? + \beta?); (\beta? + \gamma?); \top$

construction follows Thompson's construction, but when we encounter a test $\phi?$ (which is not allowed in regular expressions) we generate a single accepting state marked with ϕ.

Having constructed the ϵ-NFA, we convert it into an alternating automaton; to do so, we need to remove any ϵ-transitions as these cannot feature in alternating automata.

To do so, [Faymonville and Zimmermann, 2015] constructs a transition relation for the alternating automata that features a disjunction or a conjunction over all possible ϵ-paths. This may lead to difficulties in any resulting implementation as the number of ϵ-paths can be unbounded (in cases where we have a cycle of ϵ-paths, such as $(p?)^*$). To avoid this problem, we introduce the novel notion of *critical sets* which give us a principled way of capturing only the subset of the ϵ-paths that actually need to be considered. The key intuition behind critical sets is that we only need to consider the "best case" sets of tests to match ρ for $\langle\rho\rangle\phi$, or the "worst case" for $[\rho]\phi$.

Definition 5 (Marking Sets)**.** Let s, s' be nodes in a marked ϵ-NFA $A = (\Sigma, S, s^0, R, f, m)$. $M \subseteq \text{range}(m)$ is a *marking set* from s to s' if there exists an ϵ-path π from s to s' in A, such that $m(\pi) = M$. We say that π induces M.

Definition 6 (Critical Sets)**.** Let s, s' be nodes in a marked ϵ-NFA $A = (\Sigma, S, s^0, R, f, m)$. M is a *minimal critical set* of the markings range(m) from s to s', if M is a marking set from s to s', and there does not exist $M' \subseteq \text{range}(m)$ such that $M' \subset M$ and M' is a minimal critical set from s to s'.

The *maximal critical set* is defined dually; M is a *maximal critical set* of range(m) if M is a marking set from s to s', and there does not exist $M' \subseteq \text{range}(m)$ such that $M \subset M'$ and M' is a maximal critical set from s to s'.

As an example, consider Figure 1, the marked ϵ-NFA for $(\alpha?+\beta?); (\beta?+\gamma?); \top$. Intuitively, to travel from s^0 to s', we need to pass either β or *both* α and γ. The set of minimal critical sets from s^0 to s' is therefore $\{\{\beta\}, \{\alpha, \gamma\}\}$; if β holds we do not need to test γ to reach s'. Conversely, the set of *maximal* critical sets refers to the "worst case" sets of tests to check to travel from s^0 to s'; here we have $\{\{\alpha, \beta\}, \{\alpha, \gamma\}, \{\beta, \gamma\}\}$. Notice there is no path testing all of α, β, γ, so $\{\alpha, \beta, \gamma\}$ is not a maximal critical set.

We now exploit critical sets to convert an ϵ-NFA for a given dynamic modality to an equivalent alternating automaton. We can achieve this by constructing a suitable transition relation for each state. In particular, we explain how to use the minimal critical sets to construct a transition relation equivalent to that in [Faymonville and Zimmermann, 2015], which was shown to be correct. We show this for the diamond case; the box case is dual. Recall that the transition relations for states of the constructed alternating automaton in [Faymonville and Zimmermann, 2015] for $\langle\rho\rangle\phi$ are defined as follows:

$$R(s, A) = \begin{cases} R_\phi(s, A) & s \in S_\phi \\ R_j(s, A) & s \in S_j, j \in \{1, \ldots, k\} \\ D_{Remain} \vee D_{Leave} & s \in S_r \end{cases}$$

where $D_{Remain} = \bigvee_{s' \in S_r \setminus \{f_r\}} \bigvee_{\pi \in \Pi(s,s')} P_\pi$, $P_\pi = \bigvee_{t \in R_r(s',A)} \left(t \wedge Q_{m(\pi)}\right)$, $Q_M = \bigwedge_{\theta_j \in M} R_j(s^0_j, A)$ and $D_{Leave} = \bigvee_{\pi \in \Pi(s,f_r)} \left(R_\phi(s^0_\phi, A) \wedge Q_{m(\pi)}\right)$.

Theorem 1. Let π_1 be an ϵ-path from s to s' in a marked ϵ-NFA $A = (\Sigma, S, s^0, \rho, f, m)$, for the regular expression ρ. Let M_1 be the marking set induced by π_1 and suppose M_1 is not a minimal critical set. The transition relation for the alternating automaton for $\langle\rho\rangle\phi$ without considering π_1 is the same as that considering π_1.

Proof. We first show this for D_{Remain}. Consider P_{π_1}. Since M_1 is not a minimal critical set, by definition there is some minimal critical set $M_* \subset M_1$. We partition M_1 into M_* and $M_x = M_1 \setminus M_*$. Note that M_* itself is induced by some ϵ-path π_* from s to s' from Definition 6. The inner disjunct of D_{Remain}, $\bigvee_{\pi \in \Pi(s,s')} P_\pi$, includes both P_{π_1} and P_{π_*}. Consider that $P_{\pi_1} \vee P_{\pi_*} =$

$$\bigvee_{t \in R_r(s',A)} (t \wedge Q_{M_*} \wedge Q_{M_x}) \vee \bigvee_{t \in R_r(s',A)} (t \wedge Q_{M_*})$$

$$= \bigvee_{t \in R_r(s',A)} ((t \wedge Q_{M_*} \wedge Q_{M_x}) \vee (t \wedge Q_{M_*}))$$

$$= \bigvee_{t \in R_r(s',A)} (t \wedge Q_{M_*}) = P_{\pi_*}$$

with the second last step holding by the absorption law. Thus, $\bigvee_{\pi \in \Pi(s,s')} P_\pi = \bigvee_{\pi \in \Pi(s,s') \setminus \{\pi_1\}} P_\pi$. The proof for D_{Leave} is similar; the other cases for s are independent of $\Pi(s, s')$.

Given the theorem above, we have $D_{Remain} = \bigvee_{s' \in S_r \setminus \{f_r\}} \bigvee_{M \in MinCS(s,s')} \bigvee_{t \in R_r(s',A)} (t \wedge Q_M)$, where $MinCS(s, s')$ denotes the set of minimal critical sets from s to s'. Similarly we have $D_{Leave} = \bigvee_{M \in MinCS(s,s')} \left(R_\phi(s^0_\phi, A) \wedge Q_M\right)$. For example, for $\langle\rho\rangle\phi$ in Figure 1, we only need to consider the paths $\pi = (1, 3, 4, 5, 6, 8, 9)$ and $\pi' = (1, 2, 4, 5, 7, 8, 9)$ as opposed to all four ϵ-paths between nodes 1 and 9.

For the $[\rho]\phi$ case, we can similarly reformulate $D_{Remain} = \bigwedge_{s' \in S_r \setminus \{f_r\}} \bigwedge_{M \in MaxCS(s,s')} \bigwedge_{t \in R_r(s',A)} (t \vee Q_M)$ and $D_{Leave} = \bigwedge_{M \in MaxCS(s,s')} \left(R_\phi(s^0_\phi, A) \vee Q_M\right)$.

Our approach eagerly constructs the alternating automata, and the number of critical sets is potentially super-polynomial. Hence, we may require more than polynomial space. However, this is acceptable for our purposes, as we intend to use this algorithm for symbolic model checking with OBDDs (which themselves already use exponential space). We believe the empirical efficiency that BDDs provide outweighs this disadvantage in practice.

Computing Critical Sets. To compute the set of critical sets between states in an ϵ-NFA we introduce Algorithm 2, which can be seen as a variant of breadth-first search (BFS) over the ϵ-closure of each state in the alternating automaton.

Algorithm 2 Minimal Critical Set Finding Algorithm

```
INPUT: $\epsilon$-NFA $A$, state $s$ in $A$
OUTPUT: $MinCS(s, s')$ for each $s'$ in the $\epsilon$-closure of $s$
 1: Let $C$ be a map of states to sets of sets of tests.
 2: Let $Q$ be a queue of pairs of states and sets of tests.
 3: $Q.add((s, \{\}))$
 4: while $Q$ is not empty do
 5:     $(s', T_{s'}) = Q.dequeue()$
 6:     if $C[s'] = null$ then $C[s'] = \{T_{s'}\}$
 7:     else if $\exists x.x \in C[s'] \wedge x \subseteq T_{s'}$ then
 8:         continue                    $\triangleright$ skip line 12, in particular
 9:     else if $\exists x.x \in C[s'] \wedge T_{s'} \subset x$ then
10:         $C[s'] = (C[s'] \cup \{T_{s'}\}) \setminus \{x\}$
11:     else $C[s'] = (C[s'] \cup \{T_{s'}\})$
12:     for all $\epsilon$-transitions $(s', t)$ do $Q.add(t, T_{s'} \cup m(t))$
13: return $C$
```

We maintain $C[s']$, a cache of sets of tests we know we can use to reach the node s' so far. At each step in the BFS, if $C[s']$ contains a subset of the current test set $T_{s'}$, then we know the current path no longer leads to a (distinct) minimal critical set, so we prune it. The maximal case is analogous.

We now show Algorithm 2 returns $MinCS(s, s')$.

Theorem 2. Algorithm 2 returns $MinCS(s, s')$ for each s' in the ϵ-closure of s.

Proof. (sketch) We first observe that $C[s']$ for any s' only contains marking sets. This is apparent since only line 12 builds the actual sets, which follows the marking function.

Algorithm 2 cannot skip any minimal critical sets. Firstly, observe that a minimal critical set cannot be removed (line 10), because there is no marking set that is a subset of it. Now suppose that a minimal critical set M was not added. It must be induced by a path π we did not fully traverse (line 7). There must be some node s'' where it stopped, and some path π' using fewer tests to get to s''. However, the path composed of π' from s to s'' and π from s'' to s' must use M (as M is minimal). This path must be considered or dropped; if dropped, there must be yet another path using M. Since our ϵ-NFAs are finite, this cannot repeat indefinitely. We must consider some path using M, so M is added, a contradiction.

Furthermore, no other sets are returned – it is apparent that $C[s']$ for any s' never contains a pair of sets where one is a strict subset of the other, and any other marking set would have as a strict subset some minimal critical set.

Finally, we show our approach is fixed parameter tractable.

Theorem 3. Model checking interpreted systems against LDLK specifications by using Algorithm 1 requires time linear in the size of the model ($|G|$) and exponential in the size of the formula ($|\phi|$).

Proof. Step 1 requires $O(|\phi|)$ recursive runs of Algorithm 1 on formulae known to be smaller. Following ideas from Theorem 6.4 of [Emerson and Lei, 1985], this will not affect the overall complexity (which is clearly superadditive).

Step 2 requires polynomially many runs of Algorithm 2 and building the quantification. The runtime of Algorithm 2 is $O(2^{|\phi| \log |\phi|})$ since there are $O(2^{|\phi|})$ possible sets and it is apparent no set is ever added from the same edge twice. Building the quantification for all states involves iterating over each critical set a linear number of times; again, $O(2^{|\phi| \log |\phi|})$.

Step 3 requires time exponential in the size of the alternating automaton [Bloem *et al.*, 2006]. This automaton has size linear in $|\phi|$, so this step is exponential in $|\phi|$.

Step 4 requires time linear in $|G|$ and of the Büchi automaton (which is exponential in $|\phi|$) [Burch *et al.*, 1990]. Hence, all steps run in time linear in $|G|$, and exponential in $|\phi|$.

Note that, while we will show in Section 4 that Algorithm 2 leads to attractive experimental results due to the fact it is amenable to symbolic encoding, it is actually not optimal.

Theorem 4. Model checking interpreted systems against LDLK specifications is PSPACE-complete.

Proof. Hardness follows from PSPACE-completeness of LDL model checking [Faymonville and Zimmermann, 2015]. For membership, the translation above involves model checking polynomially many LDL formulae (requiring polynomial space) and evaluating polynomially many epistemic operators (requiring polynomial space under observational semantics). We can store the labels for polynomially many atomic propositions introduced; again these are of polynomial size.

4 Implementation and Experimental Results

We implemented the algorithms introduced in Section 3 on top of version 1.2.2 of MCMAS [Lomuscio *et al.*, 2015], an open-source BDD-based symbolic model checker. The sources and executables of the resulting toolkit, called MCMAS$_{LDLK}$, are available from [MCMAS$_{LDLK}$, 2017].

For usability reasons, MCMAS$_{LDLK}$ supports the standard propositional abbreviations and the box operator. Given an input formula ϕ, MCMAS$_{LDLK}$ first transforms $\neg\phi$ to NNF. We then build the alternating automaton, representing states explicitly but transitions symbolically. We associate each state with a BDD variable; this is motivated by the fact that the number of states is polynomial in the formula size. When processing LDL modalities, we explicitly construct ϵ-NFAs, find critical sets, and convert them to alternating automata, using the construction earlier presented.

MCMAS does not use alternating automata. We thus implemented the symbolic breakpoint construction of [Bloem *et al.*, 2006] to convert our alternating automaton into an equivalent symbolic Büchi automaton. We were able to reuse MCMAS's implementation for CTLK with fairness (by checking $EG\top$ with suitable fairness constraints), to find states Q for which there exists a path satisfying $\neg\phi$. Following LDLK semantics, Q', the complement of Q, constitutes the set of states satisfying ϕ. We thus check if ϕ holds in all initial states, by checking if $p \rightarrow q$ is valid (where p holds in I and q in Q').

In addition to the model checking algorithms, we also implemented counterexample generation. We also inherit fairness constraint support from MCMAS; while LDLK can capture them, we expect native support to be more efficient.

Performance Optimisations. MCMAS$_{LDLK}$ employs several optimisations. First, note that the checker's performance is affected by the size of the alternating automata constructed. To mitigate this MCMAS$_{LDLK}$ employs:
propositional shortcircuiting (PS): instead of constructing the corresponding alternating automaton, subformulae free of dynamic modalities and states where they hold are identified;

	time (s)				BDD memory (MB)		
N	reach	CTLK	LDLK	unop.	CTLK	LDLK	unop.
3	0.004	0.001	0.004	0.091	9.34	9.52	14.49
4	0.014	0.002	0.007	29.635	9.56	9.60	60.73
5	0.034	0.005	0.011	>86400	9.96	10.35	–
6	0.071	0.005	0.012	–	10.64	10.99	–
8	0.283	0.012	0.031	–	11.93	11.97	–
10	0.802	0.026	0.066	–	12.99	13.42	–
20	37.300	0.262	0.615	–	58.01	58.09	–
30	286.829	0.980	2.472	–	59.98	60.09	–
40	1854.067	3.216	7.865	–	167.50	167.60	–
50	4081.914	12.667	27.984	–	270.99	271.04	–

Table 1: Comparison between MCMAS and MCMAS$_{LDLK}$.

and fresh atomic propositions for these are introduced.
reachability pruning (RP): Since ϵ-NFA states with no inbound non-ϵ transitions are never transitioned to in alternating automata, these are not included in the alternating automata (but are still considered in the ϵ-NFAs to find critical sets).

The automata remain polynomial (so the complexity is unaffected), but we found these optimisations useful in practice.

We further optimised MCMAS$_{LDLK}$ for *efficiently computing conjuncts* (ECC) for the symbolic breakpoint construction, which involves computing $T_I \wedge T_{LC} \wedge T_R$ (notation from [Bloem *et al.*, 2006]). We initially computed each term separately, but found it cheaper to first compute a BDD T_R, and then use it as an accumulator, taking conjunctions with each term of T_I and T_{LC}. We believe this is because our original approach involves conjunctions of large BDDs (which has complexity in the product of BDD sizes [Bryant, 1986]).

Experimental Setup. We report below experiments run on virtual machines with two 2.70GHz CPUs and 16 GB of RAM running Ubuntu v15.10 (Linux kernel v4.2). As a testbed we took the prisoners and lightbulb scenario [Ditmarsch *et al.*, 2010]. We compared MCMAS$_{LDLK}$'s performance against MCMAS's for specifications expressible in both CTLK and LDLK, and evaluated the scalability of the approach on suitable LDLK specifications with the number of prisoners N. We report results for both optimised and unoptimised versions of MCMAS$_{LDLK}$.

Comparison with MCMAS. We verified ϕ_0 (CTLK) with MCMAS and ϕ_0' (LDLK) with MCMAS$_{LDLK}$; it can be checked that the two formulas are equivalent.

$$\phi_0 = AG\left(announce \rightarrow K_0\left(\bigwedge_{i=0}^{N-1} interrogated_i\right)\right)$$

$$\phi_0' = [\top^*]\left(announce \rightarrow K_0\left(\bigwedge_{i=0}^{N-1} interrogated_i\right)\right)$$

In both cases, agent 0 is the counter agent. Both specifications hold for all N, expressing that the counter knows that all prisoners have been interrogated whenever he announces this. The results are reported in Table 1.

Observe that, while the CTLK algorithm performs better than the optimised LDLK algorithm, both algorithms require time on the same order of magnitude. This is in sharp contrast with the underlying complexity of the corresponding decision problems. Further observe that the runtime is dominated by the computations for reachability making the difference in the performance for labelling practically insignificant.

The experiments also show that the optimisations above are essential for the scalability of the tool. In the cases presented PS was particularly significant as the expression within the box modality can be treated as a single atomic proposition.

	time (s)			BDD memory (MB)	
N	reach	LDLK	unop.	LDLK	unop.
3	0.004	0.017	0.069	10.24	13.75
4	0.014	0.031	0.087	10.58	14.20
5	0.034	0.108	0.244	12.49	19.09
6	0.071	0.115	0.205	12.52	18.29
8	0.283	0.376	0.572	15.59	35.50
10	0.802	1.091	1.500	20.38	46.84
20	37.300	23.197	23.943	58.36	58.94
30	286.829	310.658	323.992	60.09	60.35
40	1854.067	1127.992	1067.446	167.57	167.97
50	4081.914	7618.838	7866.579	271.40	271.94

Table 2: Verification results for MCMAS$_{LDLK}$ on ϕ_1.

Scalability assessment. We now report the results obtained when verifying the same scenario against the property

$$\phi_1 = \neg\langle(\neg on; \neg on; on; on)^*\rangle K_0 release$$

ϕ_1 holds iff on all paths the lightbulb is not repeatedly toggled until the counter knows the prisoners are released. In the scenario we can realise such a path by alternating the selection of a fresh non-counter with the counter; so ϕ_1 is false for any N. The results obtained are in Table 2.

As before, we observe that the LDLK algorithm appears able to scale reasonably well with the size of the model (for $N = 50$, we have 2.19×10^{25} reachable states). The optimisations implemented provide a small performance improvement (largely due to RP and ECC). The case $N = 40$ is an exception, probably owing to optimisation overheads that in this case perform poorly. Further experiments, not here discussed but included in the software package [MCMAS$_{LDLK}$, 2017], demonstrate that MCMAS$_{LDLK}$ scales well with large models but not so with long formulae. This is to be expected given Theorem 3 and, for similar reasons, it is also a feature of LTL model checkers. This is not normally considered to be too problematic since specifications are typically rather short.

5 Conclusions

As stated in the Introduction, LDL has recently been put forward as a powerful specification language for distributed systems and AI systems. A key attractiveness of LDL is that while it has the same expressivity as ω-regular languages, the complexity of the associated model checking problem is the same as LTL.

In this paper we have augmented LDL with epistemic operators and provided a semantics of the logic in terms of interpreted systems. The key contribution of the paper rests in the formulation of a model checking procedure for LDLK that is amenable to symbolic implementation. In fact, while the complexity of the model checking problem for LDL is known, no symbolic procedure, nor implementation is available. The resulting algorithm, based on the notion of critical sets for ϵ-NFA, was implemented as MCMAS$_{LDLK}$, an extension of the open-source model checker MCMAS. Our testing and the experimental results showed that MCMAS$_{LDLK}$ correctly supports LDLK, offers attractive performance and has little overhead on the fragments of LDLK also expressible in CTLK (already supported by MCMAS). To achieve efficiency, particular effort was devoted to several optimisations to handle the automata and OBDDs during verification.

In future work we intend to extend the results here presented to support finite traces, which is the object of recent work in the area [De Giacomo and Vardi, 2015].

References

[Bloem *et al.*, 2006] R. Bloem, A. Cimatti, I. Pill, M. Roveri, and S. Semprini. Symbolic implementation of alternating automata. In *International Conference on Implementation and Application of Automata*, pages 208–218. Springer, 2006.

[Bryant, 1986] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transaction on Computers*, 35(8):677–691, 1986.

[Burch *et al.*, 1990] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Sequential circuit verification using symbolic model checking. In *Design Automation Conference, 1990. Proceedings., 27th ACM/IEEE*, pages 46–51. IEEE, 1990.

[Cermák *et al.*, 2014] P. Cermák, A. Lomuscio, F. Mogavero, and A. Murano. MCMAS-SLK: A model checker for the verification of strategy logic specifications. In *Proceedings of the 26th International Conference on Computer Aided Verification (CAV14)*, volume 8559 of *Lecture Notes in Computer Science*, pages 525–532. Springer, 2014.

[Cimatti *et al.*, 1999] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: A new symbolic model verifier. In *Proceedings of the 11th International Computer Aided Verification Conference*, pages 495–499, 1999.

[Cimatti *et al.*, 2006] A. Cimatti, M. Roveri, S. Semprini, and S. Tonetta. From PSL to NBA: a modular symbolic encoding. In *2006 Formal Methods in Computer Aided Design*, pages 125–133. IEEE, 2006.

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

[De Giacomo and Vardi, 2016] G. De Giacomo and M. Y. Vardi. LTL_f and LDL_f synthesis under partial observability. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI16)*, pages 1044–1050. IJCAI/AAAI Press, 2016.

[De Giacomo *et al.*, 2014] G. De Giacomo, R. De Masellis, M. Grasso, F. M. Maggi, and M. Montali. Monitoring business metaconstraints based on LTL and LDL for finite traces. In *Proceedings of the 12th International Conference on Business Process Management (BPM14)*, volume 8659 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2014.

[Ditmarsch *et al.*, 2010] H. van Ditmarsch, J. van Eijck, and W. Wu. Verifying one hundred prisoners and a lightbulb. *Journal of Applied Non-Classical Logics*, 20(3):173–191, 2010.

[Emerson and Lei, 1985] E. A. Emerson and C. L. Lei. Modalities for model checking: Branching time logic strikes back. In *Proceedings of the 12th ACM Symposium on Principles of Programming Languages*, pages 84–96, 1985.

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

[Faymonville and Zimmermann, 2015] P. Faymonville and M. Zimmermann. Parametric linear dynamic logic (full version). *arXiv preprint arXiv:1504.03880*, 2015.

[Fischer and Ladner, 1977] M. J. Fischer and R. E. Ladner. Propositional modal logic of programs (extended abstract). In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing (STOC77)*, pages 286–294. ACM, 1977.

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

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

[Kacprzak *et al.*, 2008] M. Kacprzak, W. Nabialek, A. Niewiadomski, W. Penczek, A. Pólrola, M. Szreter, B. Woźna, and A. Zbrzezny. Verics 2007 - a model checker for knowledge and real-time. *Fundamenta Informaticae*, 85(1):313–328, 2008.

[Lomuscio *et al.*, 2015] A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: A model checker for the verification of multi-agent systems. *Software Tools for Technology Transfer*, 2015. http://dx.doi.org/10.1007/s10009-015-0378-x.

[MCMAS_{LDLK}, 2017] MCMAS_{LDLK}. . http://vas.doc.ic.ac.uk/, 2017.

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

[Pnueli, 1977] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th International Symposium Foundations of Computer Science (FOCS77)*, pages 46–57, 1977.

[Raimondi and Lomuscio, 2004] F. Raimondi and A. Lomuscio. Automatic verification of deontic interpreted systems by model checking via OBDDs. In *Proceedings of the Sixteenth European Conference on Artificial Intelligence (ECAI04)*, pages 53–57. IOS Press, 2004.

[Thompson, 1968] Ken Thompson. Programming techniques: Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422, 1968.

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

[Wolper, 1981] P. Wolper. Temporal logic can be more expressive. In *22nd Annual Symposium on Foundations of Computer Science (SFCS81)*, pages 340–348. IEEE, 1981.