A Lazy Approach to Temporal Epistemic Logic Model Checking

Alessandro Cimatti
Fondazione Bruno Kessler
Trento, Italy
cimatti@fbk.eu

Marco Gario
Fondazione Bruno Kessler
Trento, Italy
gario@fbk.eu

Stefano Tonetta
Fondazione Bruno Kessler
Trento, Italy
tonettas@fbk.eu

ABSTRACT

Temporal Epistemic Logic is used to reason about the evolution of knowledge over time. A notable example is the temporal epistemic logic KL_1, which is used to model what a reasoner can infer about the state of a dynamic system by using available observations. Applications of KL_1 span from security (verification of cryptography protocols and information flow) to diagnostic systems (fault detection and diagnosability).

In this paper, we tackle the verification of KL_1 properties under observational semantics, by proposing an effective approach that is able to deal with both finite and infinite state systems. The denotation of the epistemic atoms is computed in a lazy way, driven by the counter-examples obtained from model checking an abstraction of the property. We analyze the approach on a comprehensive set of finite- and infinite-state benchmarks from the literature, evaluate the effectiveness of various optimizations, and demonstrate that our approach outperforms existing approaches.

Keywords

Temporal Epistemic Logic; KL_1; Model Checking; Infinite State; IC3

1. INTRODUCTION

Temporal Epistemic Logic (TEL) is a modal logic combining operators for the evolution of time, and the representation of knowledge. TEL is gaining interest from the practical standpoint. Several works [10, 22, 1, 8] use temporal epistemic logic to model and analyze the knowledge of a perfect reasoner (or agent) that can observe (or sense) a part of a running system. The most common example is in the context of information security [1], or cryptographic protocols verification [8], where we are interested in guaranteeing that some information will remain private, even if some public information is shared. Another interesting example comes from the domain of diagnostic system design [10]. In this domain, we are interested in showing that a diagnoser can detect the occurrence of some non-observable situation (e.g., a fault) by only using the information provided by the available sensors. If the faults are monitored by multiple diagnosers, possibly sharing the available sensors, we obtain a multi-agent characterization of the problem, in which we can study whether one agent is able to detect more faults than another. In several works [19, 30, 10, 24, 4, 23, 4], in these domains, properties are expressed in KL_1 [21], which is an extension of LTL [27] including the knowledge operator K_A, with the restriction that no nesting of knowledge operators occurs.

To properly capture the target domains of cryptograpthic protocols and physical systems, infinite state models are needed. At the same time, infinite state model-checking for transition systems has become a consolidated area of research in the formal verification community [17, 6, 13], where efficient SAT/SMT [2] based algorithms have been developed. Although, in the general case, this is an undecidable problem, these algorithms (e.g., IC3 [15]) are able, in practice, to deal with infinite state models. Moreover, when these techniques are applied to finite state models, they usually exhibit better scalability than BDD-based [12] algorithms.

Our goal is to show how to exploit existing verification algorithms (e.g., IC3) to model-check KL_1 over finite/infinite state transition systems.

Contribution

In this work, we propose an effective approach for KL_1 model-checking of both finite and infinite state transition systems, under observational semantics. This is the first approach to model checking of temporal epistemic logic over infinite state systems that does not rely on the abstraction of the system.

Our solution is characterized by a *lazy* approach. The computation of the denotation of epistemic subformulae of the property is not carried out up-front. Rather, we delay it as much as possible, relying on a counter-example guided abstraction refinement (CEGAR) of the property. Similarly to the lazy approach in Satisfiability Modulo Theories, the epistemic atoms are initially treated as Boolean variables, and incrementally axiomatized as a result of proving the spuriousness of the counterexamples.

This idea makes it possible to deal with infinite state systems, where it might not be possible to compute the reachable state set, and yields a unified approach for both finite and infinite state systems. Moreover, the technique is independent from the underlying model-checking algorithm, and thus we can apply modern verification techniques (e.g., IC3 [11]) and leverage the impressive advancements achieved in the formal verification community.

We propose a number of optimizations to the basic algorithm, and identify a fragment of the logic (*InvKL*$_1$) that has practical interest and for which significant improvements can be achieved. We experimentally evaluate the approach, analyzing the effectiveness of the proposed optimizations, and comparing against an *eager* version of the approach, where the characterization of the epistemic subformula is computed upfront. Additionally, we compare the approach on the benchmarks that can be processed by the competitor systems (thus limited to finite state). The evaluation demonstrates that our approach can dramatically outperform existing approaches.

Appears in: *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016), J. Thangarajah, K. Tuyls, C. Jonker, S. Marsella (eds.), May 9–13, 2016, Singapore.*

Related Work

Decidability and complexity of model checking temporal epistemic logic in the context of infinite state system, has been explored in the context of Artifact Centric Systems [5, 3]. In those works, restrictions on the system are imposed in order to be able to reduce it to a finite state one. Thus, conceptually, one could apply existing finite state techniques. Our work differs in several aspects. First, we deal with transition systems, which are a common formalism to model physical systems and programs, and are typically used as underlying semantics for higher-level design specification languages. Second, we implement and evaluate our technique against both infinite and finite examples. Finally, and maybe most importantly, we use effective and well-known techniques for infinite state model checking as a black-box. In this way, we can automatically benefit from the improvements that are occurring in the area. To the best of our knowledge, [5, 3] are limited to the theoretical aspects, and no implementation is available.

Techniques for model-checking temporal epistemic logic for linear time and observational semantics for finite state systems are mostly based on BDDs. Unfortunately, BDDs are not able to deal with industrial size designs that can easily overcome 10^{200} states. For this reason, we are interested in applying SAT/SMT based algorithms.

From the algorithmic point of view, BDD-based techniques rely on the upfront, eager computation of the denotation, i.e., the set of states that satisfy a formula. In principle, it is possible to perform the eager computation of the denotation using SAT/SMT approaches but, as shown by the experimental evaluation in Section 7, this simple approach does not scale in practice. Instead, we rely on a lazy computation of the denotation.

In terms of underlying technology, we present an approach that is independent from the underlying model-checking technique. In our experimental evaluation, we implement it on top of an IC3-based verification engine for infinite-state transition systems.

MCMAS [26] and MCK [20] represent the state of the art in model checking temporal epistemic logic, and they both mostly rely on BDD-based techniques. In terms of expressiveness, we focus on *KL*$_1$, the fragment of the logic without nesting of epistemic operators and common knowledge, while MCK and MCMAS support both arbitrary nesting and common knowledge; on the other hand, the technique proposed here deals also with infinite state systems.

A few works (including MCK) use Bounded Model Checking techniques, for fragments of the logic in which the epistemic operator does not appear negated [31]. Apart from the syntactical limitation, these approaches are incomplete, since bounded model checking is, in general, an incomplete technique.

Our work is very related also to [29]. In [29] the temporal epistemic model checking problem is reduced to temporal model checking, by introducing (expressions on) variables local to an agent that are satisfied if and only if the corresponding epistemic expression is satisfied. In that work, however, the identification of such variables, called local propositions, is performed manually and up-front. Our approach provides an effective and fully automated technique to obtain an expression over the observable variables that characterizes the epistemic expression; additionally, we show that our approach works, in practice, also for infinite state transition systems.

Structure of the paper

The paper is structured as follows. Section 2 provides some necessary background. Section 3 formalizes the logic *KL*, the fragment *InvKL*$_1$, and defines the problem of model checking *KL* on a transition system. Section 4 presents an eager approach at solving the problem, while Section 5 presents the basic version of our lazy approach. Optimizations are discussed in Section 6. In Section 7 we provide a detailed experimental analysis of the approach. Section 8 concludes with directions for future work.

2. BACKGROUND

Our setting is quantifier-free first order logic. We use the standard notions of theory, satisfiability, validity, logical consequence. We denote formulas with $\phi, \varphi, \psi, I, T$, variables with v, x, y, and sets of variables with V. We refer to 0-arity predicates as Boolean variables, and to 0-arity uninterpreted functions as (theory) variables. If $V_1, \ldots, V_n$ are a sets of variables and φ is a formula, we might write $\varphi(V_1, \ldots, V_n)$ to indicate that all the variables occurring in φ are elements of $\bigcup_{i=1}^{n} V_i$. For each variable x, we assume that there exists a corresponding variable x' (the *primed version* of x). If V is a set of variables, V' is the set obtained by replacing each element x with its primed version ($V' = \{x' \mid x \in V\}$). Given an assignment s to variables in V, we denote with s' the assignment to the variables V' such that $s'(v') = s(v)$ for every $v \in V$. Given a formula φ, φ' is the formula obtained by adding a prime to each variable occurring in φ. Given a theory $\mathcal{T}$, we write $\varphi \models_{\mathcal{T}} \psi$ (or simply $\varphi \models \psi$) to denote that the formula ψ is a logical consequence of φ in the theory $\mathcal{T}$. Given a finite set V of variables with a (potentially) infinite domain, we denote with $\Sigma(V)$ the set of assignments to V.

A *transition system* (TS) S is a tuple $S = (V, I, T)$, where V is a set of (state) variables, $I(V)$ is a formula representing the initial states, and $T(V, V')$ is a formula representing the transitions. A state $s \in \Sigma(V)$ of S is an assignment to the variables V. A [finite] trace σ of S is an infinite [resp., finite] sequence of states $\sigma = s_0, s_1, \cdots$ such that $s_0 \models I$ and $\forall i.\ (s_i, s'_{i+1}) \models T$. We use $\sigma[i]$ to denote the state s_i of a trace, while we use σ^i to denote the prefix $\sigma[0], \sigma[1], \cdots, \sigma[i]$ of σ. A state s is reachable iff there is a finite trace σ such that $\sigma[i] = s$ for some i. We use $Reach$ to denote the set of reachable states. Without loss of generality, we assume the system to be deadlock free, i.e., that for all $s \in Reach$, there exists s' such that $s, s' \models T(V, V')$.

The infinite characterization of these systems is given by the infinite domain of the variables, i.e., in each state we have a finite number of variables that can potentially have an infinite domain. For example, we can have integer or rational values, and use the theory of arithmetic [18] to define the transition relation.

Given a transition system S and a property φ expressed, for example, in LTL [27], the model-checking problem asks whether all the traces of S satisfy φ, i.e., $S \models \varphi$. Let S be a transition system and let U be a set of parameters, we define the *parametric transition system* $P = (V, U, I_U, T_U)$, where $I(V, U)$ and $T(V, U, V')$ are now defined on both the state variables and parameters. Given a valuation for the parameters ($\gamma \in \Sigma(U)$), and a formula ψ we denote by $\gamma(\psi)$ the formula obtained by substituting in ψ every occurrence of u with $\gamma(u)$ for every parameter u in U. Given a parametric transition system P and a valuation for the parameters γ, we can compute the *induced* transition system, by replacing the parameters with their valuation: $P_\gamma = (V, \gamma(I_U), \gamma(T_U))$. Given an LTL property ϕ expressed over the state variables and parameters of a parametric transition system P, the *parameter synthesis* problem consists in finding the set of assignments to the parameters, called parameter region ω, s.t. the property is satisfied by every trace of the induced system, formally: $\omega = \{\gamma \mid P_\gamma \models \gamma(\phi)\}$

In this work, we rely on off-the-shelf tools for LTL model-checking and parameter synthesis for infinite-state systems [13].

3. MODEL CHECKING KL$_1$

Let A be an agent from a set $\mathcal{A}\}$ of possible agents. We want to talk about the possible knowledge of A assuming that it is a perfect reasoner. A *partially observable transition system* (POTS) is a transition system equipped with a set of observable variables $O \subseteq V$, i.e., $M = (V, I, T, O)$. We associate to each observer A a set of observable variables of the POTS M: $O_A \subseteq O$. Given an observer, we define the observation function $obs_A : \Sigma(V) \to \Sigma(O_A)$ as the projection on the observable variables of A for a given state. A *KL* formula has the following syntax:

$$\varphi := p \mid \varphi \wedge \varphi \mid \neg\varphi \mid X\varphi \mid Y\varphi \mid \varphi U \varphi \mid \varphi S \varphi \mid K_A\varphi$$

where p is a predicate, and $A \in \mathcal{A}\}$ is an observer. We define the Boolean operators $\vee, \to$, and the temporal operators globally (G), finally (F), once (O) and historically (H) as usual [25]. When dealing with only one observer, we simply write $K\varphi$.

The semantics of *KL* is defined recursively on points of traces and observation functions of the system M. Given a trace $\sigma = s_0, \cdots, s_n, \cdots$ of M, and $n \geq 0$ we have that:

- $(M, \sigma, n) \models p$ iff $\sigma[n] \models p$
- $(M, \sigma, n) \models \neg\varphi$ iff $(M, \sigma, n) \not\models \varphi$
- $(M, \sigma, n) \models X\varphi$ iff $(M, \sigma, n+1) \models \varphi$
- $(M, \sigma, n) \models Y\varphi$ iff $n > 0$ and $(M, \sigma, n-1) \models \varphi$
- $(M, \sigma, n) \models \varphi U \psi$ iff there exists $i \geq n$ such that $(M, \sigma, i) \models \psi$ and for all j, $n \leq j < i$, $(M, \sigma, j) \models \varphi$
- $(M, \sigma, n) \models \varphi S \psi$ iff there exists $i \leq n$ such that $(M, \sigma, i) \models \psi$ and for all j, $i < j \leq n$, $(M, \sigma, j) \models \varphi$
- $(M, \sigma, n) \models K_A\varphi$ iff for all traces σ' of M and integers $m \geq 0$ s.t. $obs_A(\sigma[n]) = obs_A(\sigma'[m])$ it holds that $(M, \sigma', m) \models \varphi$.

We say that a system M satisfies a formula φ ($M \models \varphi$) if every trace of M satisfies φ. The semantics that we use for K_A is called *observational semantics*. To evaluate the validity of this formula, the observer can use only the information available in the current time-point (i.e., no-memory). Although this looks quite limited, the observer can reason about past and future behaviors of the system, since it knows the model of the system. For example, imagine a counter from 0 to 10; when the observer sees the value of the counter being 10, it knows that previously it was 5. Another important reason for using observational semantics is the possibility of encoding *bounded recall*. If the observer can remember the last w observations, we just extend the system with a queue of observations, in which we store the last w observations.

Due to the observational semantics, if $(M, \sigma, n) \models K_A\varphi$, then $(M, \sigma', m) \models K_A\varphi$ for every trace σ' of M such that $obs_A(\sigma[n]) = obs_A(\sigma'[m])$. In this sense, the satisfaction of the formula $K_A\varphi$ in (σ, n) depends only on the state $\sigma[n]$. Thus, we define the denotation of $K_A\varphi$ in M (written $[\![K_A\varphi]\!]_M$) as:

$$[\![K_A\varphi]\!]_M = \{\overline{s} \in Reach \mid \forall\sigma, \forall n.\ obs_A(\sigma[n]) = obs_A(\overline{s}) \\ \Rightarrow (M, \sigma, n) \models K_A\varphi\}.$$

So, for every σ, for every $n \geq 0$, $(M, \sigma, n) \models K_A\varphi$ iff $\sigma[n] \in [\![K_A\varphi]\!]_M$. *KL*$_n$ is the syntactic fragment of *KL* in which the deepest nesting of K operators contains at most n K's: *KL*$_1$ is the fragment of *KL* without nesting of K, and *KL*$_0$ is LTL. We allow multiple agents, and thus we have potentially many K operators (multi-modal). For example, given the observers A and B, the formula $K_A p \wedge K_B \neg p$ is in *KL*$_1$, while the formula $K_A K_B p$ is not, since there is a nesting of the epistemic operator.

We call *epistemic invariants* of *KL*$_1$ (*InvKL*$_1$) the properties that fall into the following syntactic fragment (ϕ):

$$\phi := G\psi\,, \quad \psi := p \mid \psi \wedge \psi \mid \neg\psi \mid K_A\gamma$$
$$\gamma := p \mid \gamma \wedge \gamma \mid \neg\gamma \mid X\gamma \mid Y\gamma \mid \gamma U \gamma \mid \gamma S \gamma$$

Notice that, apart from the top-level G, all other temporal operators can occur only within the K (i.e., γ). *KL*$_1$ (and *InvKL*$_1$) are widely used fragments, despite their simplicity. The following are just a few examples of properties in the literature that are not only in *KL*$_1$ but in *InvKL*$_1$:

- Muddy Children [19]:
$$G(((K_i muddy_i) \vee (K_i \neg muddy_i)) \to says_i)$$
- Dining Cryptographers [30]:
$$G\big([(K_1 \neg paid_1) \wedge (K_1 \neg paid_2) \wedge (K_1 \neg paid_3)] \vee$$
$$[K_1(paid_1 \vee paid_2 \vee paid_3) \wedge \neg(K_1 paid_2) \wedge \neg(K_1 paid_3)]\big)$$
- FDI maximality [10]: $G(KOfault \to Alarm)$
- Card Games [24]: $G(allred \to K_{Player1} F(win_1))$

and more examples include the Faulty Train Gate Controller [4], the Gossip Protocol [4], and goals in planning problems [23].

Voters Example

A group of people are called to express a vote. The vote is represented by an unbounded integer value bigger than 0. The vote is secret, but the jury can access the sum of all votes (observable). This model can be captured by the following transition system:

$$\begin{aligned} Vars &: \{guess \in \mathbb{N}, vote_i \in \mathbb{N}, voted \in \mathbb{B}\} \\ Init &: \neg voted \\ Trans &: (result' = vote_0 + \cdots + vote_n) \wedge \\ &\qquad voted' \wedge guess' = guess \wedge vote_i' = vote_i \end{aligned}$$

Can the jury know what somebody voted? More formally, can the jury know what the first voter voted, i.e., if $M \models G(voted \to \neg K_{\{jury\}} vote_1 = guess)$, where $guess$ is an integer variable, and $jury = \{result, guess\}$ is the set of observable variables for the jury. Intuitively, for every possible guess, it is not possible for the jury to know that the first voter's vote matches the guess. The counter-example to this specification is a corner case: if everybody votes 0, the jury knows what everybody voted.

4. EAGER ALGORITHM

A simple way of solving the model-checking problem for *KL*$_1$, consists in computing the denotation of the epistemic atoms upfront. By replacing the epistemic atoms with their denotation, the *KL*$_1$ property can then be reduced to an LTL property. This is similar to what is done by BDD-based techniques, and we call it *eager* approach. We use the definition of denotation to compute it. In particular, we first compute the set of reachable states of the system, and then partition it based on observations. Finally, we consider only those sets of states with the same observations s.t. all of them satisfy β (e.g., for $[\![K_A\beta]\!]$). Given the set of reachable states, the denotation can be symbolically computed by performing quantifier-elimination.

A similar approach to obtain the denotation can be defined as a parameter synthesis problem [14]. The intuition is that the parameters are indicators of which states belong to the denotation. We experimented with this idea by building it on top of an IC3-based parameter synthesis engine. However, this yielded limited scalability (more details in Section 7). The intuition behind the limited performances is the following. Consider the property $G(K_A\beta \rightarrow \alpha)$ (every-time that the observer knows β, then α holds). To disprove this property, eager approaches need to compute the denotation of $K_A\beta$ and then intersect it with the denotation of $\neg\alpha$. The intersection might represent only a small set of states. Therefore, a lot of the computation performed up-front might not be needed (e.g., if α is always true). Moreover, in the case of infinite state systems, it might not be possible to represent the set of states of the denotation. For this reason, we develop the lazy approach, in which we compute only an approximation of the denotation that is sufficient to verify the property.

5. LAZY ALGORITHM

For a system to violate a property, we need a counter-example trace. We model-check an abstract version of the property, in which we treat all epistemic subformulas as propositional atoms. If we find a counter-example for this abstract property, we need to verify whether the counter-example satisfies all epistemic subformulas or whether it is spurious. If it is not spurious, we are done and the property is not satisfied. Otherwise, we need to refine our abstracted property, by learning additional constraints, and repeat. This flow is similar to the typical CEGAR [16] loop, with the significant difference that we do not refine the model but the property. The approach is divided into four main phases (Figure 1):

1. KL_1 to LTL abstraction
2. LTL Verification
3. Spuriousness check
4. Property Refinement.

Property Abstraction

For every epistemic subformula $K_A\beta$, we introduce a fresh *placeholder variable* $\rho_{K_A\beta}$ and obtain the abstracted property $\varphi_\rho = \varphi[K_A\beta/\rho_{K_A\beta}]$ by replacing each epistemic formula with the corresponding placeholder (Line 2 – BOOL_ABSTRACTION). The system is extended by adding the placeholder variables, that are initially unconstrained (Line 3). This corresponds to the most general abstraction of the property: in any state the placeholder can be true or false. Counter-examples to the abstract property, represents assignments for the epistemic subformulas that can violate the property. For example, if a state of a counter-example contains a placeholder set to true, it means that in that state we want the epistemic subformula to hold. After this step, we have a property φ_ρ that is purely LTL, and an extended transition system (M_ρ) that contains all the variables of M plus the (unconstrained) placeholder variables.

LTL Verification

The main loop (Line 4) of the algorithm checks whether $M_\rho \models \varphi_\rho$. If we verify the abstract property on the system, then also the original property is satisfied. However, the converse is not true, due to spurious counter-examples, i.e., a counter-example trace that is not consistent from the epistemic point of view. For example, there is a state where the epistemic subformula holds, but that does not

```
1:  function VERIFY($M$, $\varphi$)
2:      $\varphi_\rho$, placeholders := BOOL_ABSTRACTION($\varphi$)
3:      $M_\rho$ := EXTEND($M$, placeholders)
4:      loop
5:          cex := $M_\rho \models \varphi_\rho$
6:          if not cex then
7:              return "Satisfied"
8:          end if
9:          if IS_SPURIOUS(M, cex, placeholders) then
10:             $\varphi_\rho$ := LEARN_LEMMA(M, cex, placeholders, $\varphi_\rho$)
11:         else
12:             return cex
13:         end if
14:     end loop
15: end function
16:
17: function IS_SPURIOUS(M, cex, placeholders)
18:     for state $\in$ cex do
19:         for $\rho_{K_A\beta} \in$ placeholders do
20:             p_value := $\rho_{K_A\beta}$(state)
21:             if not ((state $\in [\![K_A\beta]\!]$) $\leftrightarrow$ p_value) then
22:                 return True // Spurious!
23:             end if
24:         end for
25:     end for
26:     return False
27: end function
```

Figure 1: Lazy Algorithm Pseudo-Code

belong to the denotation of the epistemic formula. We iterate until a valid counter-example is found, or the property is shown to hold ($M_\rho \models \varphi_\rho$). If no counter-example is found (Lines 6-7), then the model satisfies the abstracted property, and therefore we can terminate: on some systems we are able to terminate without ever checking the epistemic part. If a counter-example exists, we need to check whether it is spurious (Line 9). If this is the case, we can exclude the counter-example, otherwise we have found a valid counter example.

Spuriousness Check and Refinement

To check the spuriousness of a counter-example, we need to check that each state of the counter-example satisfies the epistemic part. This is implemented in the function IS_SPURIOUS (Line 17). Each state can be checked in isolation because the transition relation is independent from the epistemic part. For each state of the counter-example, and for each epistemic subformula (Lines 18-19), we extract the value of the placeholder (p_value, Line 20), and check whether the state belongs to the denotation. If p_value is true but the state does not belong to the denotation or, viceversa, p_value is false and the state belongs to the denotation, then we are in a spurious state, and we can exclude the counter-example. If we validated all epistemic formulas in all the states, then the counter-example is a real counter-example (Line 26).

To know whether a state s belongs to $[\![K_A\beta]\!]$, ($s \in [\![K_A\beta]\!]$) we perform the following model-checking query:

$$M \models G(\bigwedge_{x \in O_A} x = s(x) \rightarrow \beta)$$

intuitively, we ask whether each reachable state that has the same observation of s, satisfies the formula β. If M satisfies the property (positive case), we learn the lemma $\bigwedge_{x\in O_A} x = s(x) \rightarrow \rho_{K_A\beta}$, otherwise (negative case) we learn $\bigwedge_{x\in O_A} x = s(x) \rightarrow \neg\rho_{K_A\beta}$

(these are the lemmas returned by the LEARN_LEMMA function at Line 10). These lemmas impose additional constraints between all the states with the given observation and the placeholder variables, thus characterizing the epistemic atoms. In particular, if the counter-example is spurious, then we exclude it. Lemmas are learned by updating φ_ρ (Line 12), i.e., learned lemmas (λ_i) become preconditions to to φ_ρ. Thus in each iteration i, we have $\varphi_\rho^i := \lambda_i \rightarrow \varphi_\rho^{i-1}$.

InvKL$_1$

If $\varphi \in$ *InvKL*$_1$, the property rewriting step gives us a formula $G(\psi)$ where ψ is purely propositional. Since we are assuming deadlock freedom, this encodes an invariant over reachable states and we can take advantage of this fact, by using ad-hoc reachability algorithms, instead of a full LTL algorithm. This provides us with a performance boost and, more importantly, guarantees us that the counter-example will be a finite trace ending in a state that violates ψ. During the validation of the counter-example we need to validate only the last state of the trace, thus obtaining significant speed-ups, since the validation phase does not depend anymore on the length of the counter-example traces.

Example

We apply the algorithm on the property $\varphi = G(K\beta \rightarrow \alpha)$. We first rewrite it as LTL, and introduce the placeholder variable: $\varphi_\rho = G(\rho_{K\beta} \rightarrow \alpha)$. The transition system $M = (V, I, T, O)$ is extended by adding the unconstrained placeholder variable $M_\rho = (V \cup \{\rho_{K\beta}\}, I, T, O)$. The main-loop of the procedure checks whether $M_\rho \models \varphi_\rho$. Let us assume that the answer is negative, and we get the trace:

$$(o_1, \alpha, \beta, \rho_{K\beta}), (\neg o_1, \neg\alpha, \beta, \rho_{K\beta})$$

that violates the abstract property, since the last state requires the epistemic formula to hold, but α does not hold. We check whether this is a spurious counter-example, by checking the consistency of each state of the trace. In particular, we want to know if the first state belongs to the denotation of $K\beta$, since the state has observation o_1, we verify whether : $M \models G(o_1 \rightarrow \beta)$. Let us assume that this query has a positive outcome then we check the next state, and whether $M \models (G\neg o_1 \rightarrow \beta)$. Let us assume that this query has a negative outcome then we modify our property to include this lemma: $\varphi'_\rho = (\neg o_1 \rightarrow \neg\rho_{K\beta}) \rightarrow \varphi_\rho$. Since this is a spurious state, the counter-example is considered spurious, and we need to find another counter-example. We check $M_\rho \models \varphi'_\rho$ and no counter-example is found. Thus, we conclude that $M \models G(K\beta \rightarrow \alpha)$. Notice that the property is an *InvKL*$_1$ property, therefore it would have been sufficient to only verify the last.

5.1 Correctness

The following two lemmas show that the lazy approach is correct, i.e., $M \models \varphi$ iff the lazy algorithm does not produce a counter-example:

LEMMA 1. *Given M and φ, and the associated M_ρ, φ_ρ, we have that if $M_\rho \models \varphi_\rho$ then $M \models \varphi$.*

DEFINITION 1 (SPURIOUS STATE). *Let σ_ρ be a trace of M_ρ and σ be its projection on the variables of M. The state $\sigma_\rho[n]$ is* spurious *iff: for all $K_A\beta$ occurring in φ, $(M_\rho, \sigma_\rho, n) \models \rho_{K_A\beta}$ iff $\sigma[n] \in [\![K_A\beta]\!]_M$.*

LEMMA 2. *Let σ_ρ be a trace of M_ρ and σ be its projection on the variables of M. If $(M_\rho, \sigma_\rho) \models \neg\varphi_\rho$ and, for all $n \geq 0$. $\sigma_\rho[n]$ is not spurious , then $(M, \sigma) \models \neg\varphi$.*

Lemma 2 requires that the counter-example is not spurious, and in particular, that we are able to check whether a state belongs to the denotation. Therefore, we need to prove that the model-checking query that we use to check whether a state belongs (or not) to the denotation is correct:

THEOREM 1. *Given a reachable state s of M, the following three statements are equivalent:*

1. $s \in [\![K_A\beta]\!]_M$
2. $M \models G(\bigwedge_{x \in O_A} x = s(x) \rightarrow \beta)$
3. $M \models G(\bigwedge_{x \in O_A} x = s(x) \rightarrow K_A\beta)$

PROOF. 1 $\Leftrightarrow$ 3) By the definition of *KL*$_1$, $M \models G(\bigwedge_{x \in O_A} x = s(x) \rightarrow \beta)$ iff for all σ, for all n, if $obs_A(\sigma[n]) = obs_A(s)$, then $(M, \sigma, n) \models K_A\beta$. Then, by the definition of $[\![K_A\beta]\!]_M$, the condition 1 holds iff s is reachable and 2 holds.

2 $\Rightarrow$ 3) Consider a trace σ and an integer n such that $obs_A(\sigma[n]) = obs_A(s)$. Since the condition 2 holds, then for all σ', for all n', if $obs_A(\sigma'[n']) = obs_A(s)$, then $(M, \sigma', n') \models \beta$; thus, for all σ', for all n', if $obs_A(\sigma'[n']) = obs_A(\sigma[n])$, then $(M, \sigma', n') \models \beta$. Thus, $(M, \sigma, n) \models K_A\beta$.

3 $\Rightarrow$ 2) The condition 2 follows directly from 3 and the fact that $K_A\beta \rightarrow \beta$ is a tautology for all β (Axiom of knowledge **T**). $\square$

THEOREM 2. *Given a reachable state s of M, the following three statements are equivalent:*

1. $s \notin [\![K_A\beta]\!]_M$
2. $M \not\models G(\bigwedge_{x \in O_A} x = s(x) \rightarrow \beta)$
3. $M \models G(\bigwedge_{x \in O_A} x = s(x) \rightarrow \neg K_A\beta)$

PROOF. 1 $\Leftrightarrow$ 3) follows from Theorem 1 by contraposition.

2 $\Rightarrow$ 3) Consider a trace σ and an integer n such that $obs_A(\sigma[n]) = obs_A(s)$. Since the condition 2 holds, then there exists σ' and n' such that $obs_A(\sigma'[n']) = obs_A(s)$ and $(M, \sigma', n') \not\models \beta$; thus, it is not true that for all σ', for all n', if $obs_A(\sigma'[n']) = obs_A(\sigma[n])$, then $(M, \sigma', n') \models \beta$. Thus, $(M, \sigma, n) \models \neg K_A\beta$.

3 $\Rightarrow$ 2) Suppose by contradiction that $M \models G(\bigwedge_{x \in O_A} x = s(x) \rightarrow \beta)$. Thus, by Theorem 1, $M \models G(\bigwedge_{x \in O_A} x = s(x) \rightarrow K_A\beta)$. Since s is reachable, then there exists σ and n such that $\sigma[n] = s$. Thus $(M, \sigma, n) \models K_A\beta$ and, by the condition 3, $(M, \sigma, n) \models \neg K_A\beta$, which is a contradiction. $\square$

Finally, we need to show that the lemmas that we are adding to exclude a spurious counter-example are correct, i.e., the abstract model-checking problem is still a sound over-approximation of the concrete model-checking problem. More precisely, if we use the lemma to restrict the abstract state space and the abstract model checking passes, then we can still conclude that $M \models \varphi$:

THEOREM 3. *(Positive case) Assume that $M \models G(\bigwedge_{x \in O} \overline{o}(x) = x \rightarrow \beta)$ and $M_\rho \models G(\bigwedge_{x \in O} \overline{o}(x) = x \rightarrow \rho_{K\beta}) \rightarrow \varphi_\rho$ Then $M \models \varphi$.*

PROOF. (Positive case) If $M_\rho \models G(\bigwedge_{x \in O} \overline{o}(x) = x \rightarrow \rho_{K\beta}) \rightarrow \varphi_\rho$, then i) $M \models G(\bigwedge_{x \in O} \overline{o}(x) = x \rightarrow K\beta) \rightarrow \varphi$ (by Theorem 1). If $M \models G(\bigwedge_{x \in O} \overline{o}(x) = x \rightarrow \beta)$, then ii) $M \models G(\bigwedge_{x \in O} \overline{o}(x) = x \rightarrow K\beta)$ (by Theorem 1). From i) and ii), we can deduce that $M \models \varphi$.
$\square$

THEOREM 4. *(Negative case) Assume that* $M \not\models G(\bigwedge_{x \in O} \overline{o}(x) = x \rightarrow \beta)$ *and* $M_\rho \models G(\bigwedge_{x \in O} \overline{o}(x) = x \rightarrow \neg\rho_{K\beta}) \rightarrow \varphi_\rho$ *Then* $M \models \varphi$.

PROOF. (Negative case) If $M_\rho \models G(\bigwedge_{x \in O} \overline{o}(x) = x \rightarrow \neg\rho_{K\beta}) \rightarrow \varphi_\rho$, then i) $M \models G(\bigwedge_{x \in O} \overline{o}(x) = x \rightarrow \neg K\beta) \rightarrow \varphi$ (by Theorem 1). If $M \not\models G(\bigwedge_{x \in O} \overline{o}(x) = x \rightarrow \beta)$, then ii) $M \models G(\bigwedge_{x \in O} \overline{o}(x) = x \rightarrow \neg K\beta)$ (by Theorem 2). From i) and ii), we can deduce that $M \models \varphi$. □

We mentioned that if the property falls within the *InvKL*$_1$ fragment, then it is possible to validate only the last state of the counter-example.

THEOREM 5. *Let M be a system, $G\varphi$ an InvKL$_1$ property, and M_ρ and $G\varphi_\rho$ the corresponding abstractions. If there exists a trace σ_ρ of M_ρ of size n s.t. $\sigma_\rho[n] \not\models \varphi_\rho$ and $\sigma_\rho[n]$ is not spurious, then $M \not\models G\varphi$.*

PROOF. Let σ be the projection of σ_ρ on the variables of M. From the Definition 1, we know that if $\sigma_\rho[n] \not\models \varphi_\rho$ then $\sigma[n] \not\models \varphi$. Therefore, the invariant property is violated. □

5.2 Termination

System	*InvKL*$_1$	*KL*$_1$
Finite	Complete	Complete
Infinite w/ Finite Domain Obs.	Relative Complete	Incomplete
Infinite w/ Infinite Domain Obs.	Incomplete	Incomplete

Figure 2: Completeness

Our algorithm is guaranteed to terminate on finite state systems, since there is a finite number of possible values for observations and placeholders. Model checking for infinite state systems is in general undecidable, therefore we have the problem that the internal model-checking calls might not terminate. In practice, model-checkers for infinite state systems (e.g., UPPAAL [6], SAL [17], nuXmv [13]) can terminate on particular instances or classes of models. We call *relative complete* an algorithm that terminates assuming that all the model-checking queries terminate. Our approach is relative complete for infinite state systems only if we have both finite domain observations, and finite counter-examples (as in the case of *InvKL*$_1$). Otherwise, the algorithm is incomplete, since we might need to enumerate infinitely many observations, or validate infinitely many states. Figure 2 summarizes the result. Despite the theoretical result, we will show in Section 7, that our approach is able in practice to verify many models of interest.

6. OPTIMIZATIONS

Static Learning

The placeholder variables are initially unconstrained. However, there are some facts that we can learn by looking at the property, for example, the axioms of epistemic logic. First, we know that $K\beta \rightarrow \beta$. This translates into the constraint: $\rho_{K\beta} \rightarrow \beta$. This ensures that we never need to validate a counter example in which $\rho_{K\beta}$ and $\neg\beta$. Moreover, if o is a Boolean observable for A, then $K_A o \leftrightarrow o$. Thus, we add the constraints $\rho_{Ko} \leftrightarrow o$ for each observable variable of the observer A.

Lemma Generalization

During refinement we learn something about a single observation. We generalize this to cover a bigger space of the observations, relating multiple observations to the value of the placeholder. For a state s and an observation $\overline{o}$, we generalize the lemma $\overline{o} \rightarrow \rho_{K\beta}$ into $\overline{o}_1 \vee \cdots \vee \overline{o}_n \rightarrow \rho_{K\beta}$ (similarly for the negative case).

The main technique that we use to perform the generalization is parameter synthesis. Given the observation $\overline{o}$, we remove some element, and perform the parameter synthesis starting from a partial assignment. We partition the set of observables in the ones we fix and the one we parameterize: $O = O_F \cup O_P$, $O_F \cap O_P = \emptyset$. For $K\beta$ we solve the following parameter synthesis problem:

$$\omega = \{\overline{o}' \mid M \models G((\bigwedge_{x \in O_F} \overline{o}(x) = x \wedge \bigwedge_{x \in O_P} \overline{o}'(x) = x) \rightarrow \beta)\}$$

where $\overline{o}'$ is an assignment to the O_P variables. In practice, we obtain the region ω of assignments to O_P that (toghether with $\overline{o}$) imply β. Since the region is maximal, all other assignments to O_P do not entail β. Therefore, we learn the lemma: $\overline{o} \rightarrow (\omega \leftrightarrow \rho_{K\beta})$.

Choosing the partition into O_F and O_P is an interesting point of research, which we leave as future work. We propose a simple baseline heuristic, in which we randomly select a number w of observable variables. The choice of w is also heuristic, since a small value will cause overhead without gaining much generalization, while a big value will quickly lead to too many parameters. Indeed, picking $O_P = O$ (i.e., using all observable variables as parameters) is equivalent to solving the problem using the *eager* approach. In our implementation, we pick w as the logarithm of the iterations performed so far.

Dual-Rail Encoding

Validating a long counter-example is expensive. However, not all states might need to be validated. Let us consider the property: $a \wedge Xa \wedge XXKb$. A counter-example to this might be the trace: $(a, b, \rho_{Kb}), (a, b, \neg\rho_{Kb}), (a, \neg b, \neg\rho_{Kb})$. The satisfaction of the epistemic subformula in the first two states is irrelevant. We would like the model-checker to identify those states so that we can skip them. Thus, we provide a way for the counter-example to contain *don't care* information, transforming the trace above to:

$$(a, b, -), (a, b, -), (a, \neg b, \neg\rho_{Kb})$$

This saves us from checking the first two states, and can be a significant saving when the traces are long. We use the Dual-Rail encoding [28] to encode three values: *True*, *False* and *Don't Care*. For a placeholder ρ_i we introduce the variables ρ_i^{True} and ρ_i^{False}, that are mutually exclusive. If ρ_i^{True} is true, then the placeholder is true; if ρ_i^{False} is true, then the placeholder is false; if both are false, then the placeholder has a do not care value. We then modify the IS_SPURIOUS function (Figure 1) to handle the special case in which the placeholder is set to *don't care*, by considering the epistemic subformula as satisfied.

Positive Generalization via UNSAT-Cores

For epistemic atoms encoding safety properties, we can perform a more aggressive lemma generalization in case of a positive outcome from the placeholder query. If β is a safety property, when showing that $M \models G(\overline{o} \rightarrow \beta)$ holds, an IC3-based model-checker will also provide us with an *inductive invariant* ι as witness. Since ι is an inductive invariant, $\iota \rightarrow (\overline{o} \rightarrow \beta)$. We use unsat-core extraction to obtain a subset of the observations $\overline{o}$ that make the above unsatisfiable. In fact, $\iota \wedge \overline{o} \wedge \neg\beta$ is unsatisfiable, and we obtain an unsat-core expressed over the observable variables that justifies the unsatisfiability [7]. Let us call $\overline{o}'$ such an unsat core. By definition, we have that $\iota \wedge \overline{o}' \rightarrow \beta$. Therefore, $\overline{o}'$ is a generalization of $\overline{o}$. Moreover, the unsat core extraction is a purely combinational problem, that can be efficiently handled by a SAT or SMT solver.

Voters Example

In the voters example (Section 3), there are infinitely many voting combinations that constitute a counter-example. The positive lemma UNSAT-Core generalization can help us quickly find the problem, by generating the lemma:

$$(result = 0 \wedge guess = 0) \rightarrow \rho_K$$

that justifies the counterexample. Let us assume that at least one voter did not vote 0. We rewrite the property as:

$$M \models G(voted \wedge (\bigvee_{v \in voter} vote_v \neq 0) \rightarrow \neg K_{jury} vote_1 = guess)$$

To show that no other counter-example exists, we would need to check all possible values of the votes, that are infinitely many. The negative generalization based on parameter synthesis allows us to terminate, by showing that any other voting is good.

7. EXPERIMENTAL ANALYSIS

7.1 Setup

We implemented a prototype on top of an IC3 model-checker for infinite state systems using the theory of Linear Rational Arithmetic [18]. We evaluated the scalability of the approach on infinite state models for both *KL*$_1$ and *InvKL*$_1$ properties, and compared the different optimizations to identify a competitive configuration. Finally, we compared our approach against the latest versions of two state-of-the-art model checkers: MCK [20] and MCMAS [26]. Both tools implement BDD-based techniques for model-checking temporal epistemic logic with observational semantics (including common knowledge operator, that we do not consider). Experiments were executed on a 2.5Ghz Intel Xeon CPU, with a timeout of 1 hour; tools, and benchmarks are available online[1].

Benchmarks

The *Battery-Sensor* model (described in [10]) encodes a typical subsystem found in aerospace designs, in which a set of redundant sensors are powered by a redundant power supply unit containing batteries that are modeled using real-valued variables. We study multiple properties related to faults in the system, for example:

$$G(fault\_gen_1 \rightarrow K fault\_gen_1)$$
$$G(K(gen_1.off \wedge gen_2.off) \rightarrow KX^{10}(\neg device.on))$$
$$G(fault\_psu \rightarrow FKO(fault\_psu))$$

The second benchmark is a set of *magicboxes* [9], where a ball moves through a predefined pattern defined on a bi-dimensional grid. The external observer can only perform row and column observations, where row observation do not provide information on the column (and viceversa). The reasoner needs to consider the predefined path inside the magicbox and the available row and column information to try to identify the location of the ball. Scaling the size of the magicbox enables stressing the algorithms. For each magicbox we generate also an MCK and MCMAS model and test whether it is possible to know that the ball is in a given cell: $G(target\_cell \rightarrow K target\_cell)$.

[1]`https://es.fbk.eu/people/gario/aamas16/`

The *Dining Cryptographers* is a well studied problem in temporal epistemic logic [30]. We generated instances also for MCMAS and MCK, for an increasing number of cryptographers (up to 400) and verify whether if one cryptographer paid, he knows that nobody else did: $G((done \wedge paid_1) \rightarrow K_1 \neg(\bigvee_{i \in [2..n]} paid_i))$, and whether if a cryptographer paid it can eventually know that somebody else paid: $paid_1 \rightarrow XF(K_1 paid_2)$.

7.2 Results

Optimizations Evaluation

We studied the impact of all optimizations on the complete benchmark set, and identified two main configurations. For full *KL*$_1$, we perform static learning, generalization and use the dual-rail encoding. For *InvKL*$_1$, we also perform static learning and generalization, but disable dual-rail encoding. Moreover, for *InvKL*$_1$, we only perform the validation of the last state of the trace. To study the quality of these configurations, we proceed as follows. We compare our chosen configuration (Lazy Best) against all other possible configurations (i.e., for *InvKL*$_1$, we have 16 configurations in total). For each benchmark problem, we select the best configuration that is different from Lazy Best. We call that configuration Virtual Second Best Solver (VSBS). We compare Lazy Best and VSBS in Figure 3. Lazy Best times out in only 10 instances out of 349, while the VSBS times out in 20. Generalization sometimes adds significant overhead. We also compare the strategy of validating the whole trace vs. the last state (Figure 4): the latter can significantly pay off. The exceptions are cases in which a single counter-example is sufficient to learn everything needed to prove the property.

Generalization is fundamental to be able to solve the Dining Cryptographers benchmarks. In fact (Figure 5), applying generalization we are able to solve all DC problems (up-to 400 cryptographers), while without generalization our approach times out when reaching 13 cryptographers.

For the infinite state benchmark (Battery Sensor), we considered problems with increasing bounded recall. Increasing the recall, increases the size of the model and the number of observables (Figure 6). Without any optimization, the algorithm times out with recall 6 ($\sim$80 obs. variables), while with our chosen configuration, we can verify up to recall 40 (i.e., an infinite state model with 20 Real-valued variables and more than 500 Boolean variables).

Eager Approach

We implemented the eager approach on top of an IC3-based parameter synthesis engine, and compared it against the lazy approach. In Figure 7, we can see that the eager approach does not scale when increasing the problem size, i.e., for a recall of 5 the eager approach reaches the timeout of 1 hour, while the lazy approach terminates in less than a minute.

Finite State

In the finite case, we get excellent performances when compared to MCK and MCMAS. The comparison on all finite state problems (magicboxes and dining cryptographers) for MCK and MCMAS is given in Figures 8 and 9. In many cases our approach can provide up-to two orders of magnitude improvement, and solves all the 118 instances, while MCMAS times out on 20, and MCK on 66. We highlight the results for the dining cryptographers benchmark in Figure 10. In which MCMAS is able to verify models only up-to 240 dining cryptographers (MCK is not included because it times-out at 20). The lazy approach can verify problem with 400 cryptographers in slighly more than 10 minutes.

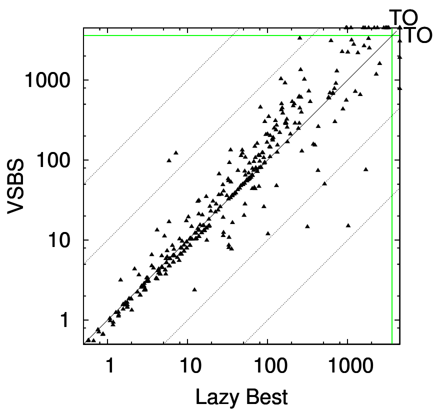

Figure 3: Comparing configurations.

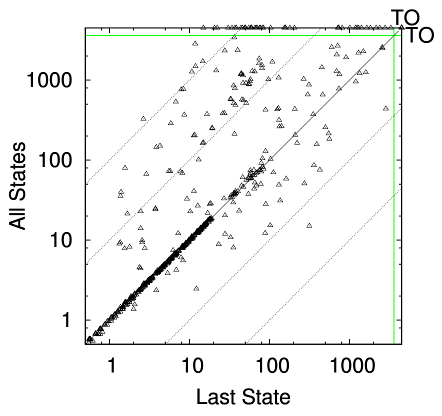

Figure 4: *InvKL*$_1$ Optimization.

#DC	Lazy w/o Gen.	Lazy w/ Gen.
3	0.26	0.15
5	1.01	0.15
7	6.07	0.13
9	58.03	0.15
11	1358.11	0.15
13	TO	0.15

Figure 5: DCs *InvKL*$_1$ properties (sec.)

Recall	#Obs	Lazy Basic	Lazy Best
0	11	3.28	1.46
5	66	3536.17	52.72
10	121	TO	103.47
20	231	TO	288.31
40	451	TO	981.11

Figure 6: Bounded Recall BS: Optimizations Impact (sec.)

Recall	Eager	Lazy
0	3.38	1.46
1	29.48	2.18
2	153.16	5.15
3	661.28	10.24
4	3028.94	13.44
5	T.O.	52.72

Figure 7: Bounded Recall BS: Eager vs Lazy (sec.)

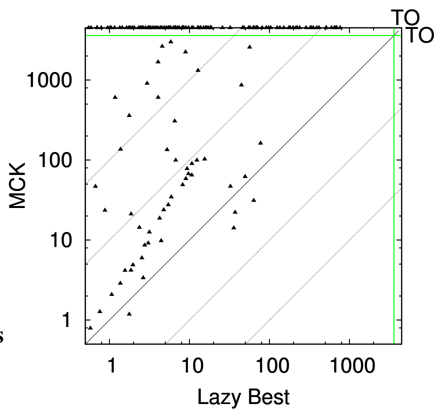

Figure 8: Lazy vs MCK (66/118 T.O.)

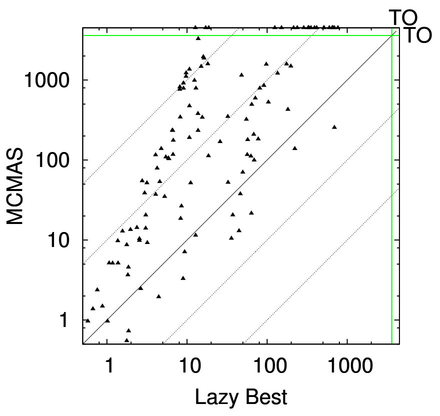

Figure 9: Lazy vs MCMAS (20/118 T.O.)

#DC	MCMAS	Lazy
40	3.66	1.83
80	26.57	8.54
120	169.43	25.9
160	322.45	55.2
200	528.42	104.02
240	1582.68	174.86
280	T.O.	287.06
320	T.O.	391.57
360	T.O.	598.4
400	T.O.	765.96

Figure 10: DCs Runtime for *KL*$_1$ properties (sec.)

8. CONCLUSIONS

KL$_1$ is a temporal epistemic logic that is extensively used in the literature. In this paper, we presented an efficient model checking approach for *KL*$_1$ under observational semantics. This is the first approach for *KL*$_1$ model checking over infinite state transition systems. The use of modern model-checking techniques, together with our lazy refinement approach and several optimizations (i.e., static learning, generalization, and dual-rail encoding) enables reasoning on models of considerable size. The experimental evaluation shows orders of magnitude improvements over existing approaches.

As future work, we plan to generalize the technique for perfect recall semantics. Moreover, it would be interesting to extend our lazy approach to reason about Artifact-Centric Systems. Finally, we would like to apply the approach on more industrial case-studies from diagnoser design.

Acknowledgements

We would like to thank Ron van der Meyden and Xiaowei Huang for providing us with an internal up-to-date version of MCK, and Franco Raimondi for providing us with a Dining Cryptographers generator for MCMAS.

REFERENCES

[1] M. Balliu, M. Dam, and G. Le Guernic. Epistemic temporal logic for information flow security. In *Proceedings of the ACM SIGPLAN 6th Workshop on Programming Languages and Analysis for Security*, PLAS '11, pages 6:1–6:12, New York, NY, USA, 2011. ACM.

[2] C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability modulo theories. *Handbook of satisfiability*, 185:825–885, 2009.

[3] F. Belardinelli, D. Grossi, and A. Lomuscio. Finite abstractions for the verification of epistemic properties in open multi-agent systems. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 854–860. AAAI Press, 2015.

[4] F. Belardinelli, A. V. Jones, and A. Lomuscio. Model checking temporal-epistemic logic using alternating tree automata. *Fundamenta Informaticae*, 112(1):19–37, 2011.

[5] F. Belardinelli, A. Lomuscio, and F. Patrizi. Verification of agent-based artifact systems. *Journal of Artificial Intelligence Research (JAIR)*, 51:333–376, 2014.

[6] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi. *UPPAAL a tool suite for automatic verification of real-time systems*. Springer, 1996.

[7] B. Bittner, M. Bozzano, A. Cimatti, M. Gario, and A. Griggio. Towards Pareto-optimal parameter synthesis for monotonic cost functions. In *Proceedings of the 14th Conference on Formal Methods in Computer-Aided Design*, pages 23–30. FMCAD Inc, 2014.

[8] I. Boureanu, M. Cohen, and A. Lomuscio. Automatic verification of temporal-epistemic properties of cryptographic protocols. *Journal of Applied Non-Classical Logics*, 19(4):463–487, 2009.

[9] M. Bozzano, A. Cimatti, M. Gario, and S. Tonetta. Formal Specification and Synthesis of FDI through an Example. In *Workshop on Principles of Diagnosis (DX'13)*, pages 174–179, 2013. Available at URL http://www.dx-2013.org/dx13-proceedings.pdf.

[10] M. Bozzano, A. Cimatti, M. Gario, and S. Tonetta. Formal design of asynchronous FDI components using temporal epistemic logic. *Logical Methods in Computer Science*, 2015.

[11] A. R. Bradley. SAT-based model checking without unrolling. In *Verification, Model Checking, and Abstract Interpretation*, pages 70–87. Springer, 2011.

[12] R. Bryant. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, 100(8):677–691, 1986.

[13] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta. The nuxmv symbolic model checker. In *Computer Aided Verification*, pages 334–342. Springer, 2014.

[14] A. Cimatti, A. Griggio, S. Mover, and S. Tonetta. Parameter synthesis with IC3. In *Formal Methods in Computer-Aided Design (FMCAD), 2013*, pages 165–168. IEEE, 2013.

[15] A. Cimatti, A. Griggio, S. Mover, and S. Tonetta. Ic3 modulo theories via implicit predicate abstraction. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 46–61. Springer, 2014.

[16] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Computer aided verification*, pages 154–169. Springer, 2000.

[17] L. De Moura, S. Owre, H. Rueß, J. Rushby, N. Shankar, M. Sorea, and A. Tiwari. Sal 2. In *Computer aided verification*, pages 496–500. Springer, 2004.

[18] B. Dutertre and L. De Moura. A fast linear-arithmetic solver for DPLL (T). In *Computer Aided Verification*, pages 81–94. Springer, 2006.

[19] R. Fagin, Y. Moses, J. Y. Halpern, and M. Y. Vardi. *Reasoning about knowledge*. MIT press, 2003.

[20] P. Gammie and R. V. D. Meyden. MCK: Model checking the logic of knowledge. *Computer Aided Verification*, pages 256–259, 2004.

[21] J. Halpern and M. Vardi. The complexity of reasoning about knowledge and time. lower bounds. *Journal of Computer and System Sciences*, 38(1):195–237, 1989.

[22] X. Huang. Diagnosability in concurrent probabilistic systems. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 853–860. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[23] F. Kominis and H. Geffner. Beliefs in multiagent planning: From one agent to many. In *Proc. ICAPS Workshop on Distributed and Multi-Agent Planning*, 2014.

[24] M. Kwiatkowska, A. Lomuscio, and H. Qu. Parallel model checking for temporal epistemic logic. In *European Conference on Artificial Intelligence*, 2010.

[25] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In R. Parikh, editor, *Logics of Programs*, volume 193, pages 196–218. Springer Berlin Heidelberg, 1985.

[26] A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: A model checker for the verification of multi-agent systems. In *Computer Aided Verification*, pages 682–688. Springer, 2009.

[27] A. Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57, 1977.

[28] J.-W. Roorda and K. Claessen. SAT-based assistance in abstraction refinement for symbolic trajectory evaluation. In *Computer Aided Verification*, pages 175–189. Springer, 2006.

[29] W. Van Der Hoek, M. Wooldridge, and S. van Otterloo. Model checking knowledge and time via local propositions: Cooperative and adversarial systems. 2004.

[30] R. Van der Meyden and K. Su. Symbolic model checking the knowledge of the dining cryptographers. In *17th IEEE Computer Security Foundations Workshop*, 2004.

[31] B. Woźna, A. Lomuscio, and W. Penczek. Bounded model checking for knowledge and real time. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 165–172. ACM, 2005.