Verification of Multi-Agent Systems via Predicate Abstraction against ATLK Specifications

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

Jakub Michliszyn
Institute of Computer Science
University of Wrocław, Poland
jmi@cs.uni.wroc.pl

ABSTRACT

We present a predicate abstraction technique for the verification of multi-agent systems against specifications defined in the epistemic logic ATLK, interpreted on a three-valued semantics. We reduce an infinite-state multi-agent program to a finite model by generating predicates automatically via SMT calls. We show that if an ATLK specification is either true or false in the abstract model, then that is also the case on the original infinite state model. We introduce and describe MCMAS$_{PA}$, a toolkit implementing the technique here described. MCMAS$_{PA}$ supports the three-valued semantics for ATLK, automatically generates program abstractions for a multi-agent system by means of automatic SMT calls, encodes the corresponding program in BDDs and reports the result. The experimental results obtained confirm that MCMAS$_{PA}$ can verify infinite-state multi-agent systems of interest.

1. INTRODUCTION

Multi-Agent Systems (MAS) are distributed computer systems where the components, or agents, rationally interact with one another and the environment in a variety of complex ways [29]. Whereas progress has been made in the theoretical analysis of MAS, both from a logical and game-theoretic perspective, we still lack comprehensive methodologies for the verification of MAS before deployment. This is of concern as techniques such as rapid prototyping, which rely on efficient evaluation, e.g., via verification, have been shown to be of significant importance in the adoption of novel paradigms in applications. In addition, the lack of robust verification methodologies also hinders the adoption of MAS in safety-critical applications.

The current state of the art for the offline analysis of MAS consists of a variety of model checking techniques including BDD-based model checking [11, 27], bounded model checking [26], and symmetry reduction [6]. Existing model checkers implement some of these in a variety of forms [24, 17, 11]. One key characteristic that these techniques and implementations have in common is that they are confined to the analysis of *finite state systems*, i.e., the models corresponding to the MAS descriptions to be analysed only have finitely many states. This is typically enforced via the input language of the toolkit by adopting variable types which can only have bounded instantiations, e.g., bounded integers. This restriction is appealing as finite state model checking is decidable, whereas infinite state model checking is in general undecidable.

While the limitation given by finite models is not significant when the MAS can be appropriately abstracted as a finite state system, most of the MAS programs that are built generate models that have infinitely many states. Additionally even if a bound for a variable can be found by analysing all possible executions, the actual range may be difficult to find for the programmer, who will generally adopt unbounded variable types.

Contribution. In this paper we begin to overcome the limitation of finite states in the practical verification of MAS. Since verifying infinite states systems is undecidable in general, the method we present is sound but not complete. Yet, we show that in some cases of interest the technique can help solving the verification problem. The method takes as input a description of an infinite state MAS and specifications in the epistemic temporal logic ATLK [2, 14]. A finite model abstracting the infinite model for the system is then generated by carrying out abstraction both on the local states and the actions of the agents. The finite model is defined by analysing the conditions that appear in the specification to be checked and in the MAS description. This analysis, which is entirely automated, results in the generation of Boolean predicates that are combined with the original MAS description to generate a finite-state MAS description. We show that if the resulting model checking problem returns either true or false, this is also the result of the verification problem for the infinite-state model.

The technique is fully automated and implemented; we present both the underlying theory for the abstraction as well as the actual predicate generation and verification. We carried out the latter by extending the open-source checker MCMAS [24] to handle the three-valued logic ATLK and implementing the algorithm for generating predicates by means of SMT calls to the SMT-checker CVC4 [4]. The resulting implementation, which we call MCMAS$_{PA}$, constitutes, to our knowledge, the first predicate abstraction toolkit for MAS.

Related Work. Epistemic versions of ATL for specifying MAS have attracted considerable attention over the past few years [14, 25, 5, 1]. The difficulty of validating systems with large state spaces has long been recognised in verification and multi-agent systems. Abstraction has been used in the context of epistemic logic [9] including in the con-

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.*

text of interpreted systems [7]. [20] presents an abstraction technique for ATL; however, it deals with finite states only, and, differently from the present contribution, it deals with complete information only.

Three-valued and multi-valued temporal logic have been developed in the context of model checking [18, 19] and abstraction methodologies have been put forward [15]. These, however, have not been extended to specifications supporting MAS and their automatisation is problematic. Predicate abstraction techniques for under- and over-approximations of infinite state systems were originally suggested in the context of temporal logic and modal μ calculus [13, 8]. While approaches use SMT engines for the generation of the finite model, they have not been extended to epistemic, nor ATL specifications.

Closer to the theoretical part of the work here presented is [21] where the theoretical underpinnings of an abstraction methodology on three valued semantics for ATLK is developed. That work, however, does not deal with infinite-state systems, no methodology for the generation of the finite models via SMT is given and, consequently, no toolkit for the actual verification is given.

Predicate abstraction techniques are normally tailored to temporal logics only. A predicate abstraction for epistemic logic has recently been proposed in [12]. While that work is also based on a three-valued logic, the semantics is different, and ATL is not supported. Indeed, [12] is tailored to the verification of service-oriented applications specified by the GSM language. In contrast we here address MAS modelled by an extension of ISPL, an existing language for reasoning about MAS. Also, a predicate abstraction technique for ATL only was discussed in [3]. The semantic setting in [3] assumes perfect recall and complete information, following the original treatment of ATL in [2]. As we remark in the next section, in principle this semantics is equivalent to the one we use in this paper; however, while [3] assume deterministic evolutions, we allow for non-deterministic transitions to account for a significant role by the environment on the system. Also, [3] is not concerned with epistemic specifications as we are here. Lastly, we are not aware of an implementation based on [3], whereas we here report on an open-source model checker for MAS supporting predicate-abstraction.

2. THE THREE-VALUED LOGIC ATLK

In this section we summarise the three-valued logic ATLK on memoryless interpreted systems with incomplete information which we adopt here. ATLK is an extension of ATL [2], the logic designed to express properties of agents' strategies, that allows also to express epistemic properties of agents and groups of agents [28]. In this context, several choices need to be made when combining the need for efficient procedures and expressive specifications. The rational for the setting we adopt here was discussed in [22].

Let $Ag = \{1, \ldots, m\}$ be a set of agents and $\mathcal{V}$ be a set of propositional variables. We use the letter Γ to denote subsets of agents, e.g., $\Gamma \subseteq Ag$; by $\overline{\Gamma}$ we denote the complementation of Γ, i.e., $Ag \setminus \Gamma$.

Definition 1 (Interpreted systems). *An* interpreted system *is a tuple* $IS = (\{L_i, Act_i, P_i, t_i\}_{i \in Ag}, I, \Pi)$ *such that:*

- *for each agent* i L_i *is a possibly infinite set of possible* local states*,* Act_i *is a set of possible* local actions*,* $P_i : L_i \to 2^{Act_i} \setminus \{\emptyset\}$ *is a* local protocol*, and* $t_i \subseteq L_i \times ACT \times L_i$ *is a* local transition relation *with* $ACT = Act_1 \times \cdots \times Act_m$*.*
- $I \subseteq L_1 \times \cdots \times L_m$ *is a set of* global initial states*,*
- $\Pi : L_1 \times \cdots \times L_m \times \mathcal{V} \to \{\mathrm{tt}, \mathrm{ff}, \mathrm{uu}\}$ *is a* labelling function*.*

Notice that the definition above allows the value of a propositional variable in a state to be unknown (uu). We will use this to represent abstract states in which the value of some propositional variables is not defined. We consider a possibly non-deterministic transition relation as this will be the case for the abstract models introduced later.

By $t.i$ we denote the i-th element of a tuple t. We define models of interpreted systems in the usual way.

Definition 2 (Models). *Given an interpreted system* $IS = (\{L_i, Act_i, P_i, t_i\}_{i \in Ag}, I, \Pi)$*, its* associated model *is a tuple* $M_{IS} = (S, T, \{\sim_i\}_{i \in Ag}, I, \Pi)$ *such that:*

- $S \subseteq L_1 \times \cdots \times L_m$ *is the set of global states reachable via* T *from the set of initial global states* I*,*
- $T \subseteq S \times ACT \times S$ *is a global transition relation such that* $T((l_1, \ldots, l_m), a, (l'_1, \ldots, l'_m))$ *iff for all* $i \in Ag$ *we have* $t_i(l_i, a, l'_i)$ *and* $a.i \in P_i(l_i)$*,*
- $\sim_i \subseteq S^2$ *is such that* $s \sim_i s'$ *iff* $s.i = s'.i$*, for all* $i \in Ag$*.*

The intended meaning of $s \sim_i s'$ is that the global states s, s' are epistemically indistinguishable for agent i [10]. For $\Gamma \subseteq Ag$, we define the relation $\sim_\Gamma$ for the common knowledge operator as the transitive closure of $(\bigcup_{i \in \Gamma} \sim_i)$.

We only consider models such that for all global states $s \in S$ and joint actions $a \in ACT$ such that $a.i \in P_i(s.i)$ for all $i \in Ag$, there exists an $s' \in S$ such that $T(s, a, s')$.

We are interested in two logics with the same syntax, but different semantics: the two-valued logic ATLK and the three-valued logic ATLK^{3v}. First, we introduce the common syntax for the logics ATLK and ATLK^{3v}.

Definition 3 (ATLK language). *The set of formulas for the logics ATLK and ATLK* ^{3v} *is defined from* $\mathcal{V}$ *by the following BNF expression:*
$\varphi ::= q \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\!\langle \Gamma \rangle\!\rangle X\varphi \mid \langle\!\langle \Gamma \rangle\!\rangle(\varphi U \varphi) \mid \langle\!\langle \Gamma \rangle\!\rangle G\varphi \mid K_i\varphi \mid C_{\Gamma'}\varphi$
where $i \in Ag$*,* $\Gamma, \Gamma' \subseteq Ag$*,* $\Gamma' \neq \emptyset$ *and* $q \in \mathcal{V}$*.*

The formulas $K_i\varphi, C_{\Gamma'}\varphi$ are read as "agent i knows that φ" and "in the group Γ' it is commonly known that φ", respectively. The reading of the ATL modalities is as follows. The formula $\langle\!\langle \Gamma \rangle\!\rangle G\varphi$ stands for "the agents in Γ *may be able* to ensure that φ holds forever"; the meaning of the "until" modality U is analogous, and $\langle\!\langle \Gamma \rangle\!\rangle X\varphi$ can be read as "the agents in Γ can ensure that φ holds at the next state irrespective of the actions of the agents in $Ag \setminus \Gamma$". We use the standard abbreviations to define $\langle\!\langle \Gamma \rangle\!\rangle F\varphi$, that stands for "the agents in Γ may be able to ensure that φ will happen at some point in the future", and the Boolean connectives.

The readings above of the ATL modalities was originally given in [25] and used in several approaches that followed. There may be applications where the readings are not appropriate; in these cases the logic CTL, which is entirely subsumed, can be used instead. Our motivation for the work

is to provide support for epistemic modalities under an efficient model checking setting; indeed, the complexity of the model checking problem in the setting we describe below is the same as that of CTLK. We refer to [22] for a discussion on these issues.

Semantics

Assume an interpreted system $IS = (\{L_i, Act_i, P_i, t_i\}_{i\in Ag}, I, \Pi)$. A (*local*, *memoryless*) *strategy* for an agent $i \in \Gamma$, or simply a strategy, is a function $f_i : s.i \to 2^{Act_i} \setminus \{\emptyset\}$ such that for each local state $s.i \in L_i$ we have $f_i(s) \subseteq P_i(s.i)$. We do not assume perfect recall, i.e., all the strategies depend on the current local state only.

Given a path $p = s_0 s_1 \ldots$, by p^i we denote s_i, the $i+1$-th element of p. Assume $\Gamma \subseteq Ag$ and an indexed set of strategies $F_\Gamma = \{f_i \mid i \in \Gamma\}$. A set of paths Y is F_Γ-*compatible* if it is a minimal non-empty set of paths such that for each path $p \in Y$, position $j \geq 0$, joint actions a, a' and state s' such that $T(p^j, a, p^{j+1})$, $T(p^j, a', s')$, and for all $i \in \Gamma$, $a.i = a'.i$ and $a.i \in f_i(s_j.i)$, there exists a path $p' \in Y$ starting with $p^0 \ldots p^j s'$. Let $out(s, F_\Gamma)$ be the family of all F_Γ-compatible sets of paths starting with s.

Assume an interpreted system $IS = (\{L_i, Act_i, P_i, t_i\}_{i\in Ag}, I, \Pi)$, its associated model $M = (S, T, \{\sim_i\}_{i\in Ag}, I, \Pi)$ and a global state $s \in S$. We inductively define the two-valued satisfaction relation $\models^2$ as follows.

- $M, s \models^2 q$ iff $\Pi(s, q) = \mathrm{tt}$,
- $M, s \models^2 \neg\varphi$ iff it is not the case that $M, s \models^2 \varphi$,
- $M, s \models^2 \varphi_1 \wedge \varphi_2$ iff $M, s \models^2 \varphi_1$ and $M, s \models^2 \varphi_2$,
- $M, s \models^2 \langle\!\langle\Gamma\rangle\!\rangle X\varphi$ iff for some strategy F_Γ, some $Y \in out(s, F_\Gamma)$ and all $p \in Y$ we have $M, p^1 \models^2 \varphi$,
- $M, s \models^2 \langle\!\langle\Gamma\rangle\!\rangle \varphi_1 U \varphi_2$ iff for some strategy F_Γ, some $Y \in out(s, F_\Gamma)$ and all $p \in Y$, there is a $k \geq 0$ s.t. we have $M, p^k \models^2 \varphi_2$ and for all $0 \leq j < k, M, p^j \models^2 \varphi_1$,
- $M, s \models^2 \langle\!\langle\Gamma\rangle\!\rangle G\varphi$ iff for some strategy F_Γ, some $Y \in out(s, F_\Gamma)$ and all $p \in Y$, $i \geq 0$ we have $M, p^i \models^2 \varphi$,
- $M, s \models^2 K_i\varphi$ iff for all $s' \sim_i s$ we have $M, s' \models^2 \varphi$,
- $M, s \models^2 C_\Gamma\varphi$ iff for all $s' \sim_\Gamma s$ we have $M, s' \models^2 \varphi$.

The definition of the three-valued satisfaction $\models^3$, introduced by [22], is given below. We assume Kleene interpretation of boolean operators, i.e., $\neg\mathrm{uu} = \mathrm{uu}$, $\neg\mathrm{tt} = \mathrm{ff}$, $\neg\mathrm{ff} = \mathrm{tt}$ and $t \wedge t'$ is tt iff both t, t' are tt, ff if any of them is ff, and uu otherwise.

- $M, s \models^3 q = \Pi(s, q)$,
- $M, s \models^3 \neg\varphi = \neg(M, s \models^3 \varphi = \mathrm{ff})$,
- $M, s \models^3 \varphi_1 \wedge \varphi_2 = (M, s \models^3 \varphi_1 = \mathrm{tt}) \wedge (M, s \models^3 \varphi_2 = \mathrm{tt})$,
- $M, s \models^3 \langle\!\langle\Gamma\rangle\!\rangle X\varphi = \mathrm{tt}$ iff for some strategy F_Γ, some $Y \in out(s, F_\Gamma)$ and all $p \in Y$, we have $M, p^1 \models^3 \varphi = \mathrm{tt}$,
- $M, s \models^3 \langle\!\langle\Gamma\rangle\!\rangle X\varphi = \mathrm{ff}$ iff for some strategy $F_{\overline{\Gamma}}$, for all $Y \in out(s, F_{\overline{\Gamma}})$ and all $p \in Y$ we have $M, p^1 \models^3 \varphi = \mathrm{ff}$,
- $M, s \models^3 \langle\!\langle\Gamma\rangle\!\rangle \varphi_1 U \varphi_2 = \mathrm{tt}$ iff for some strategy F_Γ, for some $Y \in out(s, F_\Gamma)$, and for all $p \in Y$ there is $k \geq 0$ s.t. $M, p^k \models^3 \varphi_2 = \mathrm{tt}$ and for all $j < k$, $M, p^j \models^3 \varphi_1 = \mathrm{tt}$,
- $M, s \models^3 \langle\!\langle\Gamma\rangle\!\rangle \varphi_1 U \varphi_2 = \mathrm{ff}$ iff for some strategy $F_{\overline{\Gamma}}$, for all $Y \in out(s, F_{\overline{\Gamma}})$ and for all $p \in Y$, $k \geq 0$ we have $M, p^k \models^3 \varphi_2 = \mathrm{ff}$ or there is $j < k$ s.t. $M, p^j \models^3 \varphi_1 = \mathrm{ff}$,
- $M, s \models^3 \langle\!\langle\Gamma\rangle\!\rangle G\varphi = \mathrm{tt}$ iff for some strategy F_Γ, some $Y \in out(s, F_\Gamma)$ and for all $p \in Y$, $i \geq 0$ we have $M, p^i \models^3 \varphi = \mathrm{tt}$
- $M, s \models^3 \langle\!\langle\Gamma\rangle\!\rangle G\varphi = \mathrm{ff}$ iff for some strategy $F_{\overline{\Gamma}}$, for all $Y \in out(s, F_{\overline{\Gamma}})$, and for all $p \in Y$ there is $i \geq 0$ s.t. $M, p^i \models^3 \varphi = \mathrm{ff}$
- $M, s \models^3 K_i\varphi = \mathrm{tt}$ iff $M, s' \models^3 \varphi = \mathrm{tt}$ for all $s' \sim_i s$,
- $M, s \models^3 K_i\varphi = \mathrm{ff}$ iff $M, s \models^3 \varphi = \mathrm{ff}$,
- $M, s \models^3 C_\Gamma\varphi = \mathrm{tt}$ iff $M, s' \models^3 \varphi = \mathrm{tt}$ for all $s' \sim_\Gamma s$,
- $M, s \models^3 C_\Gamma\varphi = \mathrm{ff}$ iff $M, s \models^3 \varphi = \mathrm{ff}$.

In all other cases, the value of a formula is undefined (uu).

While the conditions for the true value are very similar in both semantics, the condition for the false value is much more strict in the three valued case. Intuitively, the conditions for the ATL modalities state that "there is a strategy of agents outside of Γ to prevent a given property" rather than the weaker "there is no strategy of agents of Γ to achieve a given property". It follows that the two semantics do not coincide. For the epistemic modalities, the condition for the false value simply requires the formula to be false in the current state; such a strong condition is required to guarantee the soundness of our abstraction technique, provided later on.

An interpreted system IS satisfies a property φ, written as $IS \models^2 \varphi$, iff for all the initial states s we have $M, s \models^2 \varphi$. We define $IS \models^3 \varphi = \mathrm{tt}$ iff for all $s \in I$ we have $M, s \models^3 \varphi = \mathrm{tt}$, $IS \models^3 \varphi = \mathrm{ff}$ iff some $s \in I$ we have $M, s \models^3 \varphi = \mathrm{ff}$, and $IS \models^3 \varphi = \mathrm{uu}$ otherwise. Note that in this setting the strategies can be replaced by agents' protocols.

It is shown in [22] that for any φ we have that $IS \models^3 \varphi = \mathrm{tt}$ implies $IS \models^2 \varphi$ and $IS \models^3 \varphi = \mathrm{ff}$ implies $IS \not\models^2 \varphi$.

Remark. No technical contribution is made in this section with respect to the state of the art as we follow [22] for the semantics of the ATLK specifications. Moreover, as far as the two-valued ATL fragment is concerned, the semantics here adopted from [22] assumes that the agents are memoryless, have incomplete information of the global state. We assumed that the resulting models are non-uniform in the sense of [25]; i.e., agents can execute different actions at different global states in which their own local state is the same. This is in contrast with the original semantics for ATL [2], which stipulates complete information of the state, and perfect recall. However, it can be shown that two formulations are logically equivalent in the sense that an ATL formula is true in the setting we here adopt if and only if the formula is true in the semantics adopted in [2]. It follows that, for the two-valued fragment, an ATLK formula holds in an interpreted system under the present semantics if it holds in the ATLK logic in [16].

We follow [22] in adopting the present modelling as in applications agents are naturally assumed to have incomplete information and imperfect recall. Since our specification language includes epistemic modalities, whose definition relies on incomplete information, we find this setting more appropriate. Indeed, the modelling language we adopt in Section 3 follows this assumption, in line with other modelling approaches for MAS. However, the computations for evaluating a formula to true or false can in principle equally be performed by using the semantics in [2, 16].

3. PREDICATE ABSTRACTION OF UISPL PROGRAMS

Verification engines differ in the input language adopted to describe the system. Any abstraction methodology needs to be fine tuned to the language adopted. Here we take ISPL [24] as the starting point; other choices are possible. We chose ISPL as it constitutes the input language to MCMAS, an open-source symbolic model checker for MAS that already supports the verification of ATLK, albeit for the two-valued semantics only.

Like many of its peers, ISPL programs describe finite state systems by using bounded variable types. Below we introduce Unbounded ISPL (or UISPL for short), an extension of ISPL with unbounded integers.

Unbounded ISPL

Like ISPL, an UISPL program consists of a declaration of a number of agents (keyword `Agent`) and an environment, the definition of the initial states of the system (keyword `InitStates`), and the description of the specifications to be checked (keywords `Evaluation`, `Groups` and `Formulae`).

An agent definition consists of the following components: local states, defined by a number of variables (`Vars`) of type Boolean, enumeration, and unbounded integer; local actions (`Actions`), used to interact with other agents and the environment; a local protocol (`Protocol`), representing the actions enabled in a given state; and a local evolution function (`Evolution`), encoding the change to the local state on the basis of the current local state and the action performed by all the agents. Non-deterministic protocols are encoded in UISPL either by specifying more than one target action or by giving different actions for overlapping conditions on local states. The initial states of the system are defined by means of a Boolean formula on the agents' variables, e.g., $Ag1.pos = 1$ *or* $(Ag1.pos < 17$ *and* $Ag2.ready = true)$. Notice that every UISPL program can be naturally associated with its infinite-state interpreted system.

The specifications to be checked are given as a list of ATLK formulas, followed by the declaration for the propositions and groups of agents appearing in the specifications.

Abstract models of UISPL programs

It is easy to show that in general verifying an UISPL program is undecidable, even if the specification is limited to safety properties. The approach that we put forward is sound but not complete. Differently from other approaches including [3], the predicate abstraction technique that we introduce is modular in the agents. Our choice is motivated by the need of providing an efficient methodology for verifying epistemic specifications, which depend on the local states of the individual agents.

Consider an UISPL program, its associated interpreted system $IS = (\{L_i, Act_i, P_i, t_i\}_{i\in Ag}, I, \Pi)$ and a tuple of lists of predicates $(\vec{p}_1, \ldots, \vec{p}_m)$, where $\vec{p}_i = p_i^1, \ldots, p_i^{k_i}$. Each predicate is assumed to be a condition on an agent's state, e.g., $Ag1.pos < 16$. The abstract agent i w.r.t. a list of predicates $\vec{p}_i$ is defined as a tuple $Ag_i^A = (L_i^A, Act_i, P_i^{\text{MAY}}, P_i^{\text{MUST}}, t_i^{\text{MAY}}, t_i^{\text{MUST}})$, where:

- L_i^A is the set (i.e., conjunctions) containing all the predicates from $\vec{p}_i$,
- the *may relation* t_i^{MAY} is such that $t_i^{\text{MAY}}(c, a, c')$ for abstract states c, c' and an action a iff there are local states $l, l' \in L_i$ such that l satisfies c, l' satisfies c' and $t_i(l, a, l')$,
- the *must relation* t_i^{MUST} is such that $t_i^{\text{MUST}}(c, a, c')$ for abstract states c, c' and an action a iff for each local state $l \in L_i$ satisfying c we have $t_i(l, a, l')$ for some state l' satisfying c',
- the *may protocol* P_i^{MAY} is such that $P_i^{\text{MAY}}(c)$ is the union of all $P_i(l)$ for l satisfying c,
- the *must protocol* P_i^{MUST} is such that $a \in P_i^{\text{MUST}}(c)$ iff for every state l satisfying c we have $a \in P_i(l)$ and for every c' such that $t_i(l', a, \bar{l}')$ for some l' satisfying c and $\bar{l}'$ satisfying c', there is a state $\bar{l}$ satisfying c' such that $t_i(l, a, \bar{l})$.

While the may protocol for an abstract state is simply the union of the protocols for the corresponding concrete states, the must protocol is more complex. In the must protocol, an action a is allowed in c only if for any l_1 and l_2 satisfying c, the sets of abstractions of successors of l_1 and l_2 are the same, i.e., $\{c' \mid \exists l'.t_i(l_1, a, l') \wedge l' \text{ satisfies } c'\} = \{c' \mid \exists l'.t_i(l_2, a, l') \wedge l' \text{ satisfies } c'\}$.

EXAMPLE 1. *Consider an agent* 1 *consisting of* L_1, Act_1, P_1, t_1 *defined as follows:*

- $L_1 = \{s_1, s_2, s_3, s_4\}$ *are the local states,*
- $Act_1 = \{a\}$ *is the set containing the only possible action for the agent* 1*,*
- $P_1(l) = \{a\}$ *for each* $l \in L_1$ *is the protocol that allows the agent* 1 *to use the action* a *in all the states,*
- $t_1 = \{(s_1, a, s_3), (s_1, a, s_4), (s_2, a, s_3), (s_3, a, s_3), (s_4, a, s_4)\}$ *is the transition relation.*

Assume the predicates $p_1^1 = s_1 \vee s_2$ *and* $p_1^2 = s_3$*. The set* L_1^A *consists of three states:* $c_{12} = p_1^1 \wedge \neg p_1^2, c_3 = \neg p_1^1 \wedge p_1^2$ *and* $c_4 = \neg p_1^1 \wedge \neg p_1^2$ *Notice that* $p_1^1 \wedge p_1^2$ *is not satisfiable. Then we have*

$$
\begin{aligned}
P_1^{\text{MAY}}(c) &= \{a\} \text{ for every } c \in L_1^A \\
P_1^{\text{MUST}}(c_{12}) &= \emptyset \\
P_1^{\text{MUST}}(c) &= \{a\} \text{ for every } c \in \{c_3, c_4\}
\end{aligned}
$$

$P_1^{\text{MUST}}(c_{12}) = \emptyset$ *follows from the fact that both* s_1, s_2 *satisfy* c_{12}*, but only one of them is connected via* a *to a state satisfying* c_3*.*

The transition relations are as follows:
$t_1^{\text{MAY}} = \{(c_{12}, a, c_3), (c_{12}, a, c_4), (c_3, a, c_3), (c_4, a, c_4)\}$ *and*
$t_1^{\text{MUST}} = \{(c_{12}, a, c_3), (c_3, a, c_3), (c_4, a, c_4)\}$.

A global state $s = (l_1, \ldots, l_m)$ satisfies a tuple of clauses $(c_1, \ldots, c_m)$ if $s.i$ satisfies c_i for each $i \in A$. We can now define the concept of abstract models.

DEFINITION 4 (ABSTRACT INTERPRETED SYSTEMS). *The* abstract interpreted system *of an interpreted system IS w.r.t. a tuple $(\vec{p}_1, \ldots, \vec{p}_m)$ is a tuple $IS^A = (\{Ag_i^A\}_{i \in Ag}, I^A, \Pi^A)$, where:*

- *for each i, Ag_i^A is the abstract agent i w.r.t. $\vec{p}_i$,*
- *Π^A is such that for any state b, $q \in \mathcal{V}$ and $t \in \{\mathrm{tt}, \mathrm{ff}\}$ we have $\Pi^A(b, q) = t$ iff $\Pi(s, q) = t$ for all the states s satisfying b,*
- *$I^A = \{b \mid \exists s \in I. s \text{ satisfies } b\}$.*

The model *associated to the predicate abstraction IS^A is a tuple $M_{IS}^A = (S^A, \{T_\Gamma^A\}_{\Gamma \subseteq Ag}, \{\sim_i^A\}_{i \in Ag}, I^A, \Pi^A)$, where S^A, $\{\sim_i^A\}_{i \in Ag}$, I^A, Π^A are derived as in Definition 2 and for each $\Gamma \subseteq Ag$ we have $T_\Gamma^A(b, a, b')$ iff*

- *for each $i \in \Gamma$, $a.i \in P^{\text{MUST}}(b.i)$ and $t_i^{\text{MUST}}(b.i, a, b'.i)$, and*
- *for each $i \notin \Gamma$, $a.i \in P^{\text{MAY}}(b.i)$ and $t_i^{\text{MAY}}(b.i, a, b_i')$.*

In the following we also refer to abstract interpreted systems as predicate abstractions of a given interpreted system.

So, to construct T_Γ^A we force the agents in Γ to adhere to both their must protocol and transition relation but, more liberally, we let the protocols and transitions for the agents in $Ag \setminus \Gamma$ to be selected from their respective may relations.

We now extend the notion of F_Γ-compatibility as follows. A set of paths in M^A is F_Γ-*compatible* if it is a minimal non-empty set of paths such that for each path $p \in IS^A$, position $j \geq 0$, joint actions a, a' and state s' such that $T_\Gamma(p^j, a, p^{j+1})$, $T_\Gamma(p^j, a', s')$, and for all $i \in \Gamma$, $a.i = a'.i$ and $a.i \in f_i(s_j.i)$, there exists a path $p' \in IS^A$ starting with $p^0 \ldots p^j s'$. Having defined F_Γ-compatible paths, the two-valued and three-valued semantics for the ATL modalities can be given analogously to those in Section 2.

THEOREM 5. *Assume an interpreted system IS and let IS^A be its predicate abstraction. For any ATLK property φ, $IS^A \models^3 \varphi = \mathrm{tt}$ implies $IS \models^3 \varphi = \mathrm{tt}$ and $IS^A \models^3 \varphi = \mathrm{ff}$ implies $IS \models^3 \varphi = \mathrm{ff}$.*

PROOF. Let M be the model of IS and M^A be the model of IS^A constructed as above. Consider a state s of M and let b be the state of M^A such that s satisfies b. We prove by induction on φ that $M^A, b \models^3 \varphi = \mathrm{tt}$ implies $M, s \models^3 \varphi = \mathrm{tt}$ and $M^A, b \models^3 \varphi = \mathrm{ff}$ implies $M, s \models^3 \varphi = \mathrm{ff}$. The proof follows from the construction of the abstract system.

Basic cases. Assume $\varphi = q$. If $M^A, b \models^3 q = \mathrm{tt}$, then all the states s' satisfying b are such that $M^A, s' \models^3 q = \mathrm{tt}$, and in particular $M^A, s \models^3 q = \mathrm{tt}$. Similarly, if $M^A, b \models^3 q = \mathrm{ff}$, then all the states s' satisfying b are such that $M^A, s' \models^3 q = \mathrm{ff}$, therefore $M^A, s \models^3 q = \mathrm{ff}$.

Boolean cases are straightforward and therefore omitted.

ATL cases. Assume $\varphi = \langle\!\langle \Gamma \rangle\!\rangle X\psi$. Consider the case when $M^A, b \models^3 \varphi = \mathrm{tt}$. There is a joint action a_Γ for the agents in Γ such that for any joint action $a_{\overline{\Gamma}}$ for the agents in $\overline{\Gamma}$ and any state b' such that $T_\Gamma(b, (a_\Gamma, a_{\overline{\Gamma}}), b')$ we have $M^A, b' \models^3 \psi = \mathrm{tt}$. Consider the states s, s' such that we have $T(s, (a_\Gamma, a_{\overline{\Gamma}}), s')$, and s satisfies b and s' satisfies b'. For $i \in \Gamma$, since $a.i \in P_i(b.i)$, we have $t_i^{\text{MUST}}(b.i, (a_\Gamma, a_{\overline{\Gamma}}), b'.i)$. For any $i \notin \Gamma$, we have $t_i^{\text{MAY}}(b.i, (a_\Gamma, a_{\overline{\Gamma}}), b'.i)$ by the definition of the function t_i^{MAY}. So $T_\Gamma(b, (a_\Gamma, a_{\overline{\Gamma}}), b')$ and, by the inductive assumption, $M, s' \models^3 \psi = \mathrm{tt}$ and $M, s \models^3 \varphi = \mathrm{tt}$.

If $M^A, b \models^3 \varphi = \mathrm{ff}$, then there is a joint action $a_{\overline{\Gamma}}$ for $\overline{\Gamma}$ such that for any joint action a_Γ for Γ and any state b' such that $T_{\overline{\Gamma}}(b, (a_{\overline{\Gamma}}, a_\Gamma), b')$ we have $M^A, b' \models^3 \psi = \mathrm{ff}$. Consider states s, s' such that we have $T(s, (a_{\overline{\Gamma}}, a_\Gamma), s')$, s satisfies b and s' satisfies b'. For $i \in \overline{\Gamma}$, since $a.i \in P_i(b.i)$, we have $t_i^{\text{MUST}}(b.i, (a_{\overline{\Gamma}}, a_\Gamma), b'.i)$. For any $i \notin \overline{\Gamma}$, we have $t_i^{\text{MAY}}(b.i, (a_{\overline{\Gamma}}, a_\Gamma), b'.i)$ by the definition of the function t_i^{MAY}. So $T_{\overline{\Gamma}}(b, (a_{\overline{\Gamma}}, a_\Gamma), b')$ and, by the inductive assumption, $M, s' \models^3 \psi = \mathrm{ff}$ and $M, s \models^3 \varphi = \mathrm{ff}$.

The remaining ATL cases can be shown similarly.

Epistemic cases. Consider $\varphi = K_i\psi$. Assume $M^A, b \models^3 \varphi = \mathrm{tt}$. Then for any $b' \sim_i b$, $M^A, b' \models^3 \psi = \mathrm{tt}$. Consider any state $s' \sim_i s$. By definition, there is $b^{s'}$ such that s' satisfies $b^{s'}$ and $b^{s'} \sim_i b$, and so $M^A, b^{s'} \models^3 \psi = \mathrm{tt}$, and, by inductive assumption, $M, s' \models^3 \psi = \mathrm{tt}$. Since s' was an arbitrary state, we have $M, s \models^3 K_i\psi = \mathrm{tt}$.

Assume $M^A, b \models^3 \varphi = \mathrm{ff}$. Then we have $M^A, b \models^3 \psi = \mathrm{ff}$, and, by the inductive assumption, $M, s \models^3 \psi = \mathrm{ff}$, and therefore $M, s \models^3 \varphi = \mathrm{ff}$.

The case for common knowledge is similar. □

Verification of abstract models

We now give algorithms to compute, for an IS and a given specification φ, a pair of BDDs: one representing a set of states where the value of φ is tt and one representing a set of states where the value of φ is ff (Algorithm 1).

For atomic expressions, the BDDs can be computed using an SMT solver. Observe that for each Γ, a BDD representing T_Γ can be computed in a straightforward way. This is used in the function PREIMAGE, which is then used for all the ATL modalities. The procedures for epistemic modalities are as in [22] and therefore omitted. The remaining procedures are similar to the procedures in [22], but here they work on BDDs rather than sets, and use two kinds of transition relations and protocols (this is reflected in the definition of T_Γ).

The verification process works as follows: firstly, we generate the list of predicates. This can be done in several ways; we show one in the next section. Secondly, we generate the abstract model and compute BDDs representing the sets of states where the value of the given specification is tt and ff. If the value is tt in all the initial states, then we return tt; if it is ff in some of the initial states, then we return ff; otherwise, we return uu. In the latter case the value of the specification cannot be determined on the basis of the abstraction constructed above.

4. THE MODEL CHECKER MCMAS$_{\text{PA}}$

We now describe the model checker MCMAS$_{\text{PA}}$, which extends the open-source model checker MCMAS [24] by supporting the three-valued semantics for ATLK and the predicate abstraction methodologies described above. MCMAS$_{\text{PA}}$ is implemented in C++ and uses the SMT solver CVC4 [4].

MCMAS$_{\text{PA}}$ takes as an input an infinite state MAS encoded as an UISPL program and uses the predicate abstraction technique described in the previous sections to generate an abstract, finite state system. The abstract system is then verified against the given ATLK specification by means

Algorithm 1 The model checking procedure for verifying an IS against ATLK^{3v} specifications.

procedure checkFormula(IS, φ)
  if φ is atomic **then**
    return checkAtomic(IS, φ)
  else if $\varphi = \neg\varphi'$ **then**
    $(S_{\mathrm{tt}}, S_{\mathrm{ff}}) \leftarrow$ checkFormula(IS, φ')
    return $(S_{\mathrm{ff}}, S_{\mathrm{tt}})$
  else if $\varphi = \varphi_1 \wedge \varphi_2$ **then**
    $(S^1_{\mathrm{tt}}, S^1_{\mathrm{ff}}) \leftarrow$ checkFormula(IS, φ_1)
    $(S^2_{\mathrm{tt}}, S^2_{\mathrm{ff}}) \leftarrow$ checkFormula(IS, φ_2)
    return $(S^1_{\mathrm{tt}} \cap S^2_{\mathrm{tt}}, S^1_{\mathrm{ff}} \cup S^2_{\mathrm{ff}})$
  else if $\varphi = \langle\!\langle\Gamma\rangle\!\rangle X\varphi'$ **then**
    return checkATLX(IS, Γ, φ')
  else if $\varphi = \langle\!\langle\Gamma\rangle\!\rangle(\varphi_1 U \varphi_2)$ **then**
    return checkATLU($IS, \Gamma, \varphi_1, \varphi_2$)
  else if $\varphi = \langle\!\langle\Gamma\rangle\!\rangle G\varphi'$ **then**
    return checkATLG(IS, Γ, φ')
  else
    return checkEpistemic(IS, Γ, φ)

Algorithm 2 Auxiliary procedures for checkFormula. The function Primed changes the variables in the BDD for S so they correspond to the third parameter of T_Γ.

procedure checkAtomic(IS, φ)
  $S_{\mathrm{tt}} \leftarrow \emptyset$, $S_{\mathrm{ff}} \leftarrow \emptyset$
  for each abstract state b **do**
    if $b \wedge \neg\varphi$ is unsatisfiable **then**
      $S_{\mathrm{tt}} \leftarrow S_{\mathrm{tt}} \cup \{b\}$
    else if $b \wedge \varphi$ is unsatisfiable **then**
      $S_{\mathrm{ff}} \leftarrow S_{\mathrm{ff}} \cup \{b\}$
  return $(S_{\mathrm{tt}}, S_{\mathrm{ff}})$
procedure preimage(IS, Γ, S)
  $Result = compose(T_\Gamma, Primed(S))$
  for each $i \notin \Gamma$ **do**
    $Result = \forall a_i.Result$
  for each $i \in \Gamma$ **do**
    $Result = \exists a_i.Result$
  return $Result$

Algorithm 3 Procedures for handling ATL modalities.

procedure checkATLX(IS, Γ, φ)
  $(S'_{\mathrm{tt}}, S'_{\mathrm{ff}}) \leftarrow$ checkFormula(IS, φ)
  $S_{\mathrm{tt}} \leftarrow$ preimage($IS, \Gamma, S'_{\mathrm{tt}}$)
  $S_{\mathrm{ff}} \leftarrow$ preimage($IS, \overline{\Gamma}, S'_{\mathrm{ff}}$)
  return $(S_{\mathrm{tt}}, S_{\mathrm{ff}})$
procedure checkATLU(IS, Γ, φ_1, φ_2)
  $(S^1_{\mathrm{tt}}, S^1_{\mathrm{ff}}) \leftarrow$ checkFormula(IS, φ_1)
  $(S_{\mathrm{tt}}, S_{\mathrm{ff}}) \leftarrow$ checkFormula(IS, φ_2)
  repeat
    $S'_{\mathrm{tt}} \leftarrow S_{\mathrm{tt}}$
    $S_{\mathrm{tt}} \leftarrow S_{\mathrm{tt}} \cup$ (preimage($IS, \Gamma, S_{\mathrm{tt}}$) $\cap S^1_{\mathrm{tt}}$)
  until $S_{\mathrm{tt}} = S'_{\mathrm{tt}}$
  repeat
    $S'_{\mathrm{ff}} \leftarrow S_{\mathrm{ff}}$
    $S_{\mathrm{ff}} \leftarrow S_{\mathrm{ff}} \cap$ (preimage($IS, \overline{\Gamma}, S_{\mathrm{ff}}$) $\cup S^1_{\mathrm{ff}}$)
  until $S_{\mathrm{ff}} = S'_{\mathrm{ff}}$
  return $(S_{\mathrm{tt}}, S_{\mathrm{ff}})$
procedure checkATLG(IS, Γ, φ)
  $(S'_{\mathrm{tt}}, S'_{\mathrm{ff}}) \leftarrow$ checkFormula(IS, φ)
  repeat
    $S'_{\mathrm{tt}} \leftarrow S_{\mathrm{tt}}$
    $S_{\mathrm{tt}} \leftarrow$ preimage($IS, \Gamma, S_{\mathrm{tt}}$) $\cap S_{\mathrm{tt}}$
  until $S_{\mathrm{tt}} = S'_{\mathrm{tt}}$
  repeat
    $S'_{\mathrm{ff}} \leftarrow S_{\mathrm{ff}}$
    $S_{\mathrm{ff}} \leftarrow$ preimage($IS, \Gamma, S_{\mathrm{ff}}$) $\cup S'_{\mathrm{ff}}$
  until $S_{\mathrm{ff}} = S'_{\mathrm{ff}}$
  return $(S_{\mathrm{tt}}, S_{\mathrm{ff}})$

of symbolic verification on the three-valued semantics for ATLK. If the result of this check is either true or false, by the result of the previous section, it follows that this is also the case for the original MAS.

MCMAS$_{\mathrm{PA}}$ generates a list of predicates in parallel for each agent based on the declarations in the UISPL file. As stated earlier, the technique, differently from other predicate abstraction methodologies, is modular with respect to the agents in the system. For each agent i, we build the list of predicates based on the agent's protocol, the specification to be checked, and the evaluation of the atoms in the system. Specifically, for each atomic expression in the protocol or evaluation for i, or any expression in the specifications for the atoms dependent only on i, we tentatively add this expression to the list of predicates. If the expression is not equivalent to an existing predicate or to its complement, we add the expression to the list. The SMT solver CVC4 is employed to check these equivalences. The process terminates when all the expressions have been analysed.

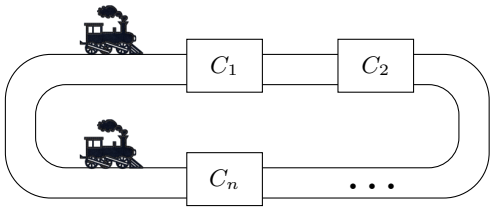
Example 2. *Consider the property*

$$Ag1.counter \leq 0 \wedge\ Ag1.v < Ag2.v \wedge \langle\!\langle\Gamma\rangle\!\rangle X Ag1.counter > 0$$

The resulting list of predicates from this formula for $Ag1$ *consists only of a single predicate* $Ag1.counter \leq 0$. *The predicate* $Ag1.counter > 0$ *is not included as it is the negation of* $Ag1.counter \leq 0$. *The predicate* $Ag1.v < Ag2.v$ *is also omitted as it refers to a different agent.*

Then, CVC4 is employed to generate the may relation. Due to the computational cost of SMT queries on several variables, MCMAS$_{\mathrm{PA}}$ employs various heuristics to reduce the computation time. First, we test whether the straightforward translations of the transitions in the UISPL program, where each condition is either replaced by the corresponding predicate or omitted, result in valid may transitions. Then, for each abstract state, we execute a query whose results represent the remaining transitions in the abstract model. This is the most time-consuming step of the

Figure 1: n tunnels version of the scenario.



verification procedure. The may protocol is computed in the straightforward way.

While the process to generate the may transition relation and protocol is relatively straightforward, to generate the must protocol, queries involving universal quantifiers are required. Solving these queries is undecidable in general; the SMT solver we employed was unable to answer any queries that would be of practical use. We therefore used an approximation and defined the must protocol on the basis of the may relation and protocol as follows. For each abstract state and action, if there is exactly one possible may transition from this state and action, we add this action and state to the must protocol. The must protocol obtained is a subset of the must protocol previously described; it follows that the preservation theorem still holds in this case. The must relation is defined accordingly.

When the abstract model has been defined, the atoms in the specifications are replaced appropriately in view of the predicates introduced and the finite-state model is verified by using the symbolic three-valued labelling procedure described in the previous section except for one exception. Instead of using the computationally expensive procedure checkAtomic, instead we check whether the given atomic condition was used to generate the list of predicates. If it was, all the states satisfying the corresponding predicate are returned; if they are not satisfying its negation, then the predicate is removed as redundant; otherwise, the empty set is returned. This does not influence the soundness of the algorithm.

5. EXPERIMENTAL EVALUATION

To evaluate the methodology we report the experimental results obtained on a infinite-state variation of the widely-studied, epistemic version of the train-gate-controller benchmark [14]. In the revised version we consider, there are two trains travelling in the opposite direction on two separate circular tracks. The tracks merge into several critical sections (tunnels) where only one train can be present at any given time. The operation at each of the tunnels is governed by a controller (see Fig. 1). At each time step each train either stays in the same section of the track or requests access to the next tunnel. If the request is granted by the controller, it moves to next part of the track. Each controller keeps track of which train has crossed its junctions more times, so that, if it chooses to do so, it may prioritise one over the other.

We modelled the scenario above in UISPL by considering an agent for each of the trains and each of the tunnel controllers. Each controller is equipped with an unbounded integer variable recording how many times each of the train has been given priority to cross the junction. Since there is no bound on this number, even the model with one tunnel has an unbounded number of states. For evaluation purposes we considered several versions of the scenario by scaling the number of the tunnels. All UISPL programs used can be accessed from [23].

To evaluate the scenario we considered the following specifications.

- Train A knows that it may never be able to leave its initial position:

$$K_{Train_a}\langle\langle Train_a\rangle\rangle G\, Train_a.pos = s_0$$

- Train A knows that together with the controller it may eventually be able to enter the first tunnel:

$$K_{Train_a}\langle\langle Train_a, Controller\rangle\rangle F Train_a.pos = s_1$$

- The controller may be able to guarantee that train A enters the tunnel at least as many times as train B:

$$\langle\langle Controller\rangle\rangle G\, Controller.counter \geq 0$$

- Train B may not be able to force Train A into the tunnel:

$$\neg\langle\langle Train_b\rangle\rangle F\, Train_a.pos = s_1$$

We ran the tool against the specifications above for different number of controllers on an Intel® Core™ i7-2600 CPU (3.40GHz, 8 cores) running Linux Kernel 3.11.0-20-generic. MCMAS$_{PA}$ generated appropriate predicates in each of these cases on the basis of the atoms presented in the specification and the UISPL programs. The resulting finite-state descriptions were verified and the specifications were found to hold, as expected. Table 1 reports the time required to generate the abstraction, and the time required to verify the abstract model. Observe that most of the computation time was devoted to computation of the abstract system rather than verification itself. The memory footprint of the checker was between 11MB and 30MB.

We found that MCMAS$_{PA}$ was unable to resolve the truth value of the specification expressing that there is a joint strategy for a controller and Train A to reach a state where the counter equals 10:

$$\langle\langle Controller, Train_a\rangle\rangle F\, Controller.counter = 10$$

This is because all the states of the controller with $counter \in \{1, \ldots, 9\}$ are abstract into a single state with no successors w.r.t. the must relation; therefore the value of the formula in the abstract model cannot be determined. As we remarked earlier the technique is sound but cannot be complete; while we may improve the technique to resolve this particular specifications, there are bound to be formulas whose value the tool will not be able to determine.

The train gate controller scenario focuses on MAS with many agents having a single integer variable. For further benchmarking we now discuss a protocol with two agents only, but which are described by several integer variables.

To do so we consider a simple protocol in which an agent S intends to communicate n integers to an agent R in the presence of a perfect communication line. To communicate the value v_i of the ith number, S uses v_i times the action $send_i$. When all the numbers are sent, S loops using the action *done* forever.

Table 1: Train Gate Controller: verification time

Tunnels	States	Abstract	Verify
1	6	0.3s	<0.1s
2	36	0.6s	0.1s
3	128	2.0s	0.14s
4	400	5.2s	0.2s
5	1152	13s	0.23s
6	3136	31s	0.3s
7	8192	78s	0.36s
8	20736	198s	0.4s
9	51200	489s	0.55s
10	123904	1477s	0.9s

Table 2: Number Transmission: verification time

Integers	States	Abstract	Verify
2	6	0.55s	<0.1s
4	56	2.29s	<0.1s
6	240	7.01s	0.11s
8	992	20.8s	0.15s
10	4032	57.2s	0.21s
12	16256	154s	0.33s
14	65280	406s	0.53s
16	261632	1031s	0.96s
18	1047552	2717s	1.9s
20	4192256	5903s	3.7s

Both agents are implemented by using n integer variables and an enumeration variable *status* that may either be equal to *busy* (representing states where the transmission is in progress), or *done* (representing the end of the protocol). The initial values of the integer variables of S are chosen randomly; the values for R's variables are initialised to 0. Both status values are initially set to *busy*. The protocols and evaluations can be implemented in UISPL in the straightforward way.

We used MCMAS$_{PA}$ to verify the following epistemic specification: whenever R's status is equal to done, then R knows that S's status is equal to done. This can be encoded in ATLK as

$$\langle\langle\emptyset\rangle\rangle G(R.status = done \Rightarrow K_R S.status = done)$$

MCMAS$_{PA}$ reported the specification to hold in all cases considered. The computation times are presented in Table 2. Similarly to the train gate controller example, the numbers of states and verification times increase exponentially with the number of integer variables. The verification time grows slightly slower than in the Train Gate Controller case; one of the reasons for this is that here we have a fixed number of possible actions, and therefore the transition relation is smaller.

We also attempted to verify the protocol's termination by checking the specification

$$\langle\langle R, S\rangle\rangle F R.status = done$$

In this case MCMAS$_{PA}$ reported the formula to be undefined. In the simplest case, with one number being transmitted, this is because the checker considers the abstraction where all the positive values of the number are abstracted into a single state. From this state the only allowed action is $send_0$, resulting in a loop. On the abstract model it therefore cannot be shown that any further state will be reached. Doing so would require refining the model, which we leave for further work.

6. CONCLUSIONS

In this paper we have proposed a technique for the verification of infinite-state MAS by means of predicate abstraction. The specification language we support is based on epistemic logic; we provide support for a weak form of ATL at no extra computational cost, but the logic CTL is fully supported by the technique.

The formalism we put forward shares some of the features present in [22], notably the three-valued approach to ATLK. Yet it differs from it in several respects. Firstly, we here employ a more general semantics that comprises a may and must relation. Secondly, the notion of state abstraction is more general here as we can collapse states with different protocols. Thirdly, [22] assumes finite-state systems and provides no details as to how generate finite models from a MAS program. In contrast in this work we have provided a methodology to do so by creating appropriate Boolean predicates by means of SMT calls and use these to define an abstract, finite-state program for the MAS.

Regarding the evaluation of the technique the checker that we released uses UISPL, an extension of MCMAS's original input language ISPL, to model MAS with unbounded integer values. We have shown that the checker can verify infinite state models automatically. We are not aware of any other model checker for MAS supporting infinite states nor employing predicate abstraction. While no comparison can be made, the experimental results obtained demonstrate the tool's performance is relatively robust with respect to the number of unbounded variables present in the program.

In future work we plan to introduce further optimisations in the generation of SMT calls so that more complex programs can be verified. We also intend to introduce a methodology for action abstraction so that particular types of MAS programs can be more effectively abstracted. The methodology that we studied here is independent of the input language; while we here defined and adopted UISPL, in the future we plan to support other input languages. We see these as steps to achieve the long-term goal to move from the verification of ad-hoc MAS models to the verification of actual MAS code.

ACKNOWLEDGEMENTS

This research was funded by the EPSRC under grant EP/ I00520X.

REFERENCES

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

[2] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.

[3] T. Ball and O. Kupferman. An abstraction-refinement framework for multi-agent systems. In *Proceedings of*

the 21st Annual IEEE Symposium on Logic in Computer Science (LICS06), pages 379–388. IEEE, 2006.

[4] C. Barrett, C. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli. CVC4. In *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV11)*, pages 171–177. Springer, 2011.

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

[6] M. Cohen, M. Dam, A. Lomuscio, and H. Qu. A data symmetry reduction technique for temporal-epistemic logic. In *Proceedings of the 7th International Symposium on Automated Technology for Verification and Analysis (ATVA09)*, volume 5799 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2009.

[7] M. Cohen, M. Dam, A. Lomuscio, and F. Russo. Abstraction in model checking multi-agent systems. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS09)*, pages 945–952. IFAAMAS Press, 2009.

[8] S. Das, D. Dill, and S. Park. Experience with predicate abstraction. In *Proceedings of the 11th International Conference on Computer Aided Verification (CAV99)*, pages 160–171. Springer, 1999.

[9] C. Enea and C. Dima. Abstractions of multi-agent systems. In *Proceedings of 5th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS07)*, volume 4696 of *Lecture Notes in Computer Science*, pages 11–21. Springer, 2007.

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

[11] 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.

[12] P. Gonzalez, A. Griesmayer, and A. Lomuscio. Verification of GSM-based artifact-centric systems by predicate abstraction. In *Proceedings of the 13th International Conference on Service Oriented Computing (ICSOC15)*, volume 9435 of *Lecture Notes in Computer Science*, pages 253–268. Springer, 2015.

[13] S. Graf and H. Saïdi. Construction of abstract state graphs with pvs. In *Proceedings of the 9th International Conference on Computer Aided Verification (CAV97)*, volume 1254 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 1997.

[14] 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.

[15] M. Huth, R. Jagadeesan, and D. Schmidt. Modal transition systems: A foundation for three-valued program analysis. *ACM Transactions on Programming Languages and Systems*, pages 155–169, 2001.

[16] W. Jamroga and W. van der Hoek. Agents that know how to play. *Fundamenta Informaticae*, 62:1–35, 2004.

[17] 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.

[18] B. Konikowska and W. Penczek. Model checking for multi-valued computation tree logics. In *Beyond Two: Theory and Applications of Multiple Valued Logic*, pages 193–210. Physica-Verlag, 2003.

[19] B. Konikowska and W. Penczek. Model checking for multivalued logic of knowledge and time. In *Proceedings of the 5th international joint conference on Autonomous Agents and Multiagent Systems (AAMAS06)*, pages 169–176. IFAAMAS, 2006.

[20] M. Köster and P. Lohmann. Abstraction for model checking modular interpreted systems over ATL. In *Programming Multi-Agent Systems*, volume 7217 of *Lecture Notes in Computer Science*, pages 95–113. Springer, 2012.

[21] A. Lomuscio and J. Michaliszyn. An abstraction technique for the verification of multi-agent systems against ATL specifications. In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR14)*, pages 428–437. AAAI Press, 2014.

[22] A. Lomuscio and J. Michaliszyn. Verifying multi-agent systems by model checking three-valued abstractions. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS15)*, pages 189–198, 2015.

[23] A. Lomuscio and J. Michaliszyn. MCMAS$_{PA}$: A predicate abstraction model checker for multi-agent systems. http://vas.doc.ic.ac.uk/software/extensions, 2016.

[24] 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.

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

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

[27] F. Raimondi and A. Lomuscio. Automatic verification of multi-agent systems by model checking via OBDDs. *Journal of Applied Logic*, 5(2):235–251, 2005.

[28] W. van der Hoek and M. Wooldridge. Model checking cooperation, knowledge, and time - a case study. *Research In Economics*, 57(3):235–265, 2003.

[29] M. Wooldridge and N. R. Jennings. Intelligent agents: theory and practice. *Knowledge Engineering Review*, 2(10):115–152, 1995.