Practical Abstractions for Model Checking Continuous-Time Multi-Agent Systems

Yan Kim
Interdisciplinary Centre for Security, Reliability, and Trust, SnT, University of Luxembourg
Esch-sur-Alzette, Luxembourg
yan.kim@uni.lu

Wojciech Jamroga
Institute of Computer Science, Polish Academy of Science
Warsaw, Poland
Interdisciplinary Centre for Security, Reliability, and Trust, SnT, University of Luxembourg
Esch-sur-Alzette, Luxembourg
w.jamroga@ipipan.waw.pl

Wojciech Penczek
Institute of Computer Science, Polish Academy of Science
Warsaw, Poland
w.penczek@ipipan.waw.pl

Laure Petrucci
LIPN, CNRS UMR 7030, Université Sorbonne Paris Nord
Villetaneuse, France
laure.petrucci@lipn.univ-paris13.fr

ABSTRACT

Model checking of temporal logics in a well established technique to verify and validate properties of multi-agent systems (MAS). However, practical model checking requires input models of manageable size. In this paper, we extend the model reduction method by variable-based abstraction, proposed recently by Jamroga and Kim, to the verification of real-time systems and properties. To this end, we define a real-time extension of MAS graphs, extend the abstraction procedure, and prove its correctness for the universal fragment of Timed Computation Tree Logic (TCTL). Besides estimating the theoretical complexity gains, we present an experimental evaluation for a simplified model of the Estonian voting system and verification using the Uppaal model checker.

KEYWORDS

model checking abstraction; real-time systems; timed automata

ACM Reference Format:
Yan Kim, Wojciech Jamroga, Wojciech Penczek, and Laure Petrucci. 2025. Practical Abstractions for Model Checking Continuous-Time Multi-Agent Systems. In *Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Detroit, Michigan, USA, May 19 – 23, 2025*, IFAAMAS, 10 pages.

1 INTRODUCTION

Temporal logics have been extensively used to formalize properties of agent systems, including reachability, liveness, safety, and fairness [26]. Moreover, temporal model checking is a popular approach to formal verification of MAS [4, 19]. However, the verification is known to be hard, both theoretically and in practice. State-space explosion is a major obstacle here, as models of real-world systems are huge and infeasible even to generate, let alone verify. In consequence, model checking of MAS w.r.t. their *modular representations* ranges from **PSPACE**-complete to undecidable [13, 53].

Much work has been done to contain the state-space explosion by smart representation and/or reduction of input models. Symbolic model checking based on SAT- or BDD-based representations of the state/transition space [36, 42, 44, 45, 47, 48, 50] fall into the former group. Model reduction methods include partial-order reduction [28, 41, 49], equivalence-based reductions [2, 6, 25], and state abstraction [21], see below for a detailed discussion.

In this paper, we extend the idea of *variable-based abstraction* [38, 39] to the verification of *real-time multi-agent systems* [1, 3, 12, 52]. Similarly to [38, 39], our abstraction operates entirely on the high-level, modular representation of an asynchronous MAS. That is, it takes a *concrete modular representation* of a MAS as input, and generates an *abstract modular representation* as output. Moreover, it produces the abstract representation without generation of the explicit state model, thus avoiding the usual computational bottleneck.

Related Work. State abstraction was introduced in [21], and studied intensively in the context of temporal verification [14, 16, 16–18, 23, 31, 54]. However, those works propose lossless abstraction that typically obtain up to an order of magnitude reduction of the state space and output models that are still too large for practical verification. Here, we focus on lossy may abstractions, based on user-defined equivalence relations [20, 22, 27, 29–32, 35, 46].

May/must abstractions for strategic properties have been investigated in [5, 8, 9, 24, 43]. In all those cases, the abstraction method is defined directly on the concrete model, i.e., it requires to first generate the concrete global states and transitions, which is exactly the bottleneck that we want to avoid. In contrast, our method operates on modular (and compact) model specifications, both for the concrete and the abstract model. Data abstraction methods for infinite-state MAS [7, 10] come close in that respect, but they still generate explicit state models. Even closer, [38, 39] proposed recently a user-friendly abstraction scheme via removal of variables in the modular agent templates. However, all the above approaches deal only with the verification of *untimed* models and properties.

Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Y. Vorobeychik, S. Das, A. Nowe (eds.), May 19 – 23, 2025, Detroit, Michigan, USA. © 2025 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

Timed models of MAS and their verification have been studied for over 30 years now, see e.g. [1, 3, 12] and especially [52] for an overview. Time-abstracting bisimulation for timed automata was studied in [56]. In this paper, we extend the ideas and results from [38] to models of real-time asynchronous MAS of [3].

The study of MAS using Uppaal was conducted in [33] and [34]. However, its authors tackled a different problem — strategy synthesis, in a different setting — stochastic timed games. In some ways, their methods can be seen as complementary to ours, as they first synthesize the (witness) strategy and then check its validity; whereas we mainly focus on the verification of safety properties.

2 REASONING ABOUT REAL-TIME MAS

We start with extending the (untimed) representations of asynchronous MAS in [38] to their timed counterparts.

We first adopt the usual definitions of clocks in timed systems, e.g. [1]. Let $\mathcal{X} = \{x_1, \ldots, x_{n_\mathcal{X}}\}$ be a finite set of clock variables. A clock valuation is a mapping $v : \mathcal{X} \mapsto \mathbb{R}_+$.[1] Given a valuation v, a delay $\delta \in \mathbb{R}_+$ and $X \subseteq \mathcal{X}$, $v + \delta$ denotes the valuation v', such that $v'(x) = v(x) + \delta$ for all $x \in \mathcal{X}$, and $v[X = 0]$ denotes the valuation v'', such that $v''(x) = 0$ for all $x \in X$ and $v''(x) = v(x)$ for all $x \in \mathcal{X} \setminus X$.

The set $C_\mathcal{X}$ of clock constraints over $\mathcal{X}$ is inductively defined by the following grammar:

$$\mathfrak{cc} ::= \top \mid x_i \sim c \mid x_i - x_j \sim c \mid \mathfrak{cc} \wedge \mathfrak{cc},$$

where $\top$ denotes the truth value, $\forall_{i,j \in \{1,\ldots,n_\mathcal{X}\}} x_i, x_j \in \mathcal{X}$, $c \in \mathbb{N}$, and $\sim \in \{<, \leq, =, \geq, >\}$.

Let $\mathcal{V} = \{v_1, \ldots, v_{n_\mathcal{V}}\}$ be a finite set of discrete (typed) numeric variables over finite domains. By $Eval_\mathcal{V}$ we denote the set of evaluations over the set of (discrete) variables $\mathcal{V}$; an evaluation $\eta \in Eval_\mathcal{V}$ maps every variable $v \in \mathcal{V}$ to a literal from their domain $\eta(v) \in domain(v)$. Expressions are constructed from variables $\mathcal{V}$ and literals $\bigcup_{i=1}^{n_\mathcal{V}} domain(v_i)$ using the arithmetic operators. Atomic formulas/conditions are built from expressions and relation symbols. They can be further combined with logical connectives to form predicates. The set of all possible predicates over the variables $\mathcal{V}$ is denoted by $Cond_\mathcal{V}$.

The sets of all valuations satisfying $\mathfrak{cc} \in C_\mathcal{X}$ and all evaluations satisfying $g \in Cond_\mathcal{V}$ shall be denoted by $[\![\mathfrak{cc}]\!]$ and $[\![g]\!]$ respectively. For $v \in \mathcal{V}$, $k \in domain(v)$, by $g[v = k]$ we denote the substitution of all occurrences of the variable v in g with the literal k; analogously, for $V \subseteq \mathcal{V}$, $K = \{k_v \in domain(v) \mid v \in V\}$, by $g[V = K]$ we denote $g[v = k_v \mid v \in \mathcal{V}]$.

For simplicity, we assume that both $\mathcal{X}$ and $\mathcal{V}$ have (some) ordering fixed. In the sequel, the terms *variables* and *clocks* will mean *discrete variables* and *continuous variables* respectively.

2.1 Multi-Agent Graphs with Clocks

In this section, we first introduce the specification of an individual agent, and a set of agents. The associated model is then defined, as well as its behaviour.

Definition 2.1 (TAG). A *timed agent graph* (TAG) is a 10-tuple $G = (\mathcal{V}, Loc, l_0, g_0, Act, Effect, Chan, \mathcal{X}, \mathcal{I}, E)$, where:

[1] By $\mathbb{R}_+$ we denote the set of non-negative real numbers.

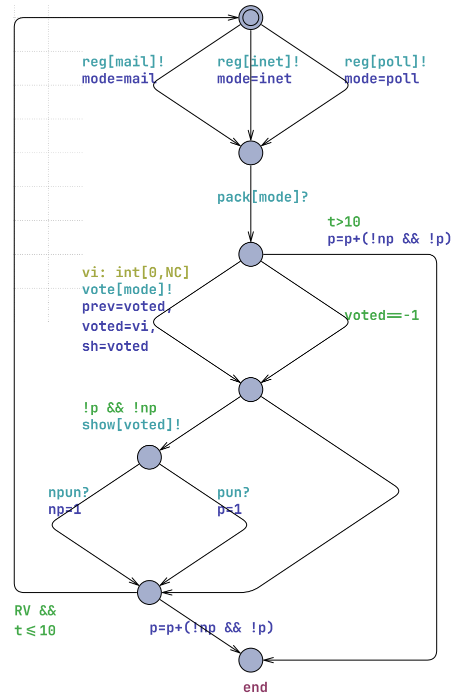

Figure 1: Timed agent graph for the Voter

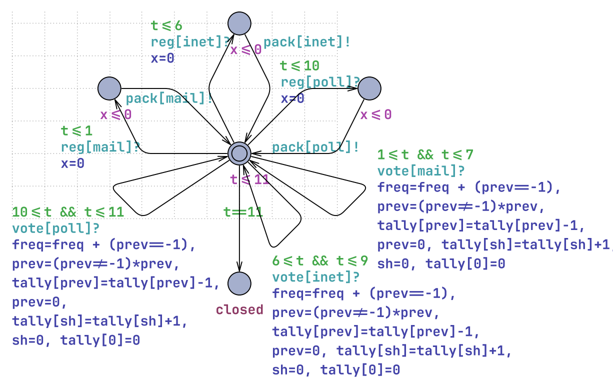

Figure 2: Timed agent graph for the Authority

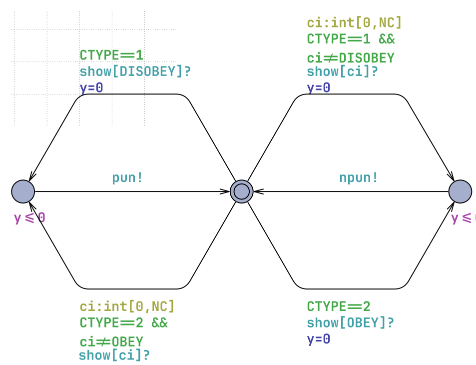

Figure 3: Timed agent graph for the Coercer

- $\mathcal{V}$ is a finite set of variables,
- *Loc* is a finite set of locations, $l_0 \in Loc$ is the initial location,
- $g_0 \in Cond_{\mathcal{V}}$ is the initial condition, s.t. $[\![g_0]\!]|_{\mathcal{V}}$ is a singleton,[2]
- *Act* is a set of actions, where $\tau \in Act$ stands for "do nothing",
- $Effect : Act \times Eval_{\mathcal{V}} \mapsto Eval_{\mathcal{V}}$ is the effect function, such that $Effect(\tau, \eta) = \eta$ for any $\eta \in Eval_{\mathcal{V}}$,
- *Chan* is a finite set of asymmetric one-to-one synchronisation channels; by *Sync* we denote the set of synchronisation labels of the form "$ch!$" and "$ch?$" for emitting and receiving on a channel $ch \in Chan$ respectively, and "$-$" for no synchronisation,
- $\mathcal{X}$ is a finite set of clocks,
- $\mathcal{I} : Loc \mapsto C_{\mathcal{X}}$ is a location invariant,
- $E \subseteq Loc \times Cond_{\mathcal{V}} \times C_{\mathcal{X}} \times Sync \times Act \times \mathcal{P}(\mathcal{X}) \times Loc$ is a finite set of labelled edges, which define the local transition relation.

Following the common practice, the notation $l \xhookrightarrow{g,\mathfrak{cc},\varsigma,\alpha,X} l'$ shall be used as a shorthand for $(l, g, \mathfrak{cc}, \varsigma, \alpha, X, l') \in E$. Given a synchronisation label ς, its complement denoted by $\overline{\varsigma}$ is defined as $\overline{ch!} = ch?$, $\overline{ch?} = ch!$ and $\overline{\varsigma} = \varsigma$ when $\varsigma = -$.

Example timed agent graphs for a voting and coercion scenario are shown in Figs. 1, 2, and 3. The scenario is explained in more detail in Section 5. In order to improve readability, the truth valued invariants, as well as the edge label components for $g = \top$, $\mathfrak{cc} = \top$, $\varsigma = -$, $X = \varnothing$, and $\alpha = \tau$ are not explicitly depicted.

Definition 2.2 (TMAS Graph). A *timed multi-agent system graph* is a multiset[3] of timed agent graphs $MG = \{\!|G^1, \ldots, G^{\mathcal{N}}|\!\}$ with distinguished[4] set of shared variables $\mathcal{V}_{sh}$. For simplicity, we assume that *MG* has (some) fixed ordering of its elements.

2.2 Models of Timed MAS Graphs

A *combined TMAS graph* merges the agent graphs in *MG* into a single agent graph whose nodes represent the possible tuples of locations in *MG*.

Definition 2.3 (Combined TMAS Graph). Given a TMAS graph $MG = \{\!|G^1, \ldots, G^{\mathcal{N}}|\!\}$ with the set of shared variables $\mathcal{V}_{sh}$, where $G^i = (\mathcal{V}^i, Loc^i, l_0^i, g_0^i, Act^i, Effect^i, Chan^i, \mathcal{X}^i, \mathcal{I}^i, E^i)$, $1 \leq i \leq \mathcal{N}$, the *combined TMAS graph* of *MG* is defined as a timed agent graph $G_{MG} = (\mathcal{V}, Loc, l_0, g_0, Act, Effect, Chan, \mathcal{X}, \mathcal{I}, E)$, where $Chan = \varnothing$, $\mathcal{V} = \bigcup_{i=1}^{\mathcal{N}} \mathcal{V}^i$, $Loc = \prod_{i=1}^{\mathcal{N}} Loc^i$, $l_0 = (l_0^1, \ldots, l_0^{\mathcal{N}})$, $\mathcal{X} = \bigcup_{i=1}^{\mathcal{N}} \mathcal{X}^i$, and $g_0 = (g_0^1 \wedge \ldots \wedge g_0^{\mathcal{N}})$. The location invariant is defined as $\mathcal{I}(l^1, \ldots, l^{\mathcal{N}}) = \mathcal{I}^1(l^1) \wedge \ldots \wedge \mathcal{I}^{\mathcal{N}}(l^{\mathcal{N}})$ and the set of actions $Act = \{\alpha^{u_1} \circ \ldots \circ \alpha^{u_k} \mid \alpha^i \in Act^i, i \in \{u_1, \ldots, u_k\} \subseteq \{1, \ldots, \mathcal{N}\}\}$, The effect function $Effect : Act \times Eval_{\mathcal{V}} \mapsto Eval_{\mathcal{V}}$ is defined by:

$$Effect(\alpha, \eta) = \begin{cases} \eta[\mathcal{V}^i = Effect^i(\alpha, \eta_{|\mathcal{V}^i})(\mathcal{V}^i)] & \text{if } \alpha \in Act^i \\ Effect(\alpha^i, Effect(\alpha^j, \eta)) & \text{if } \alpha = \alpha^i \circ \alpha^j \end{cases}$$

where $\eta[X = Y]$ denotes the evaluation η', such that $\eta'|_X = \{Y\}$ and $\eta'|_{\mathcal{V}\setminus X} = \eta|_{\mathcal{V}\setminus X}$.

Given a pair of agents G^i and G^j of distinct indices $i \neq j$ with $(l_1, g_1, \mathfrak{cc}_1, \varsigma, \alpha_1, X_1, l_1') \in E^i$, $(l_2, g_2, \mathfrak{cc}_2, \overline{\varsigma}, \alpha_2, X_2, l_2') \in E^j$, the set of labelled edges E is obtained inductively by applying the rules:

$$\frac{l_1 \xhookrightarrow{g_1,\mathfrak{cc}_1,ch!,\alpha_1,X_1} l_1',\ l_2 \xhookrightarrow{g_2,\mathfrak{cc}_2,ch?,\alpha_2,X_2} l_2'}{l_1, l_2 \xhookrightarrow{g_1 \wedge g_1,\mathfrak{cc}_1 \wedge \mathfrak{cc}_2,-,\alpha_2 \circ \alpha_1,X_1 \cup X_2} l_1', l_2'} \qquad \frac{l_1 \xhookrightarrow{g_1,\mathfrak{cc}_1,-,\alpha_1,X_1} l_1'}{l_1, l_2 \xhookrightarrow{g_1,\mathfrak{cc}_1,-,\alpha_1,X_1} l_1', l_2}$$

$$\frac{l_1 \xhookrightarrow{g_1,\mathfrak{cc}_1,ch?,\alpha_1,X_1} l_1',\ l_2 \xhookrightarrow{g_2,\mathfrak{cc}_2,ch!,\alpha_2,X_2} l_2'}{l_1, l_2 \xhookrightarrow{g_1 \wedge g_1,\mathfrak{cc}_1 \wedge \mathfrak{cc}_2,-,\alpha_1 \circ \alpha_2,X_1 \cup X_2} l_1', l_2'} \qquad \frac{l_2 \xhookrightarrow{g_2,\mathfrak{cc}_2,-,\alpha_2,X_2} l_2'}{l_1, l_2 \xhookrightarrow{g_2,\mathfrak{cc}_2,-,\alpha_2,X_2} l_1, l_2'}$$

The *model* further unfolds the combined TMAS graphs by explicitly representing the reachable valuations of model variables.

Definition 2.4 (Model). The *model* of a timed agent graph G over AP is a 5-tuple $\mathcal{M}(G) = (St, ini, \longrightarrow, AP, L)$, where:

- $St = Loc \times Eval_{\mathcal{V}} \times \mathbb{R}_+^{n_{\mathcal{X}}}$ is a set of global states,
- $ini = (l_0, \eta_0, v_0) \in St$, such that $\eta_0 \in [\![g_0]\!]$ and $\forall_{0 \leq i \leq n_{\mathcal{X}}} v_0(x_i) = 0$, is an initial global state,
- $\longrightarrow \subseteq St \times St$ is the transition relation composed of:
  - *delay-transitions*: $(l, \eta, v) \xrightarrow{\delta} (l, \eta, v + \delta)$ for $\delta \in \mathbb{R}_+$ with $v, v + \delta \in [\![\mathcal{I}(l)]\!]$,
  - *action-transitions*: $(l, \eta, v) \xrightarrow{\alpha} (l', \eta', v')$ for $l \xhookrightarrow{g,\mathfrak{cc},-,\alpha,X} l'$ with $v \in [\![\mathcal{I}(l)]\!], v' \in [\![\mathcal{I}(l')]\!], \eta \in [\![g]\!], v \in [\![\mathfrak{cc}]\!], \eta' = Effect(\alpha, \eta)$ and $v' = v[X = 0]$,
- $AP \subseteq (Cond_{\mathcal{V}} \cup Loc)$ is a finite set of atomic propositions,
- $L : St \mapsto \mathcal{P}(AP)$ is a labelling function, such that $L(l, \eta, v) \subseteq (\{g \in Cond_{\mathcal{V}} \mid \eta \in [\![g]\!]\} \cup \{l\})$.

The model $\mathcal{M}$ of a TMAS graph *MG* is given by the model of its combined TMAS graph, i.e. $\mathcal{M}(MG) = \mathcal{M}(G_{MG})$.

We now formally define paths of the model. For this paper, we consider only *progressive*[5] paths that are free from the timelocks and deadlocks. In general, this is the property that valid models of the system should have. Furthermore, such a condition can be checked both statically and dynamically (see e.g., [52]).

Definition 2.5. A *path* from the state $s_0 \in St$ of the model M is an infinite sequence of states $\pi = s_0 s_1 s_2 \ldots$, such that $s_i \in St$, $s_{2i} \xrightarrow{\delta_i} s_{2i+1} \xrightarrow{\alpha_i} s_{2i+2}$, where $\delta_i \in \mathbb{R}_+$, $\alpha_i \in Act$, for every $i \geq 0$ and $\sum_{i \in \mathbb{N}} \delta_i = \infty$. Given $i \geq 0$, by $\pi[i] = s_i$ and $\pi[i:] = s_i s_{i+1} \ldots$ we denote i-th state and i-th suffix of π respectively. An *initial path* of a model M is a path π that starts with the initial state *ini*, that is $\pi[0] = ini$. The set of all paths of M is denoted by $Paths_M$, the set of all paths starting in $s \in St$ by $Paths_M(s)$.[6]

A state $s \in St$ is *reachable* in M iff there exists an initial path $\pi \in Paths_M$, such that $\pi[i] = s$ for some $0 \leq i < \infty$. The set of all reachable states in M is denoted by $Reach(M)$.

An evaluation $\eta \in Eval_{\mathcal{V}}$ is *reachable* at $l \in Loc$ iff there is reachable state of the form $(l, \eta, v) \in Reach(M)$ for some $v \in \mathbb{R}_+^{n_{\mathcal{X}}}$.

A *local domain* is a function $d : Loc \mapsto \mathcal{P}(Eval_{\mathcal{V}})$ that maps every location $l \in Loc$ to the set of its reachable evaluations, that is $d(l) = \{\eta \in Eval_{\mathcal{V}} \mid (l, \eta, v) \in Reach(M)\}$.

---

[2] In other words, $[\![g_0]\!] \neq \varnothing$ and $(\eta_1, \eta_2 \in [\![g_0]\!]) \Rightarrow (\forall_{v \in \mathcal{V}} \eta_1(v) = \eta_2(v))$.

[3] In order to avoid any possible confusion with the ordinary sets, we shall denote a multiset container using the "$\{\!|$" and "$|\!\}$" brackets.

[4] Note that the set of shared variables must be explicitly pointed (rather than, for example, being derived from those occurring in two or more agent graphs) due to the possibility of a TMAS graph containing multiple instances of the same agent graph, which in turn may have both shared and non-shared variables.

[5] Also called *time-divergent*.

[6] When a model M is clear from the context, the subscript M is omitted.

2.3 Logical Reasoning about TMAS Graphs

We shall now define the branching-time (timed) logic **TCTL*** [11] that generalizes **TCTL** [1], **CTL** [15].

For a set of atomic propositions AP, the syntax of **TCTL*** state formulae φ and path formulae ψ is given by the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{A}\psi \mid \mathsf{E}\psi,$$

$$\psi ::= \varphi \mid \neg\psi \mid \psi \vee \psi \mid \psi\, \mathrm{U}_I\, \psi,$$

where $p \in AP$, I is an interval in $\mathbb{R}_+$ with integer bounds of the form $[a, b]$, $(a, b]$, $[a, c)$, (a, c) for $a, b \in \mathbb{N}$, $c \in \mathbb{N} \cup \{\infty\}$. The temporal operator U stands for "until", the path quantifiers A and E stand for "for all paths" and "exists a path" respectively. Intuitively, an interval I constrains the modal operator it subscribes to. Boolean connectives and additional constrained temporal operators "sometime" (denoted by F_I) and "always" (denoted by G_I) can be derived as usual. In particular, $\mathrm{F}_I\varphi \equiv \top\, \mathrm{U}_I\, \varphi$, $\mathrm{G}_I\varphi \equiv \neg\mathrm{F}_I\neg\varphi$. We will sometimes omit the subscript for $I = [0, \infty)$, writing U as a shorthand for $\mathrm{U}_{[0,\infty)}$.

From the above, we can define other important logics:

TCTL restriction of **TCTL***, where every occurrence of temporal operators is always preceded with a path quantifier,

TACTL restriction of **TCTL**, where negation can only be applied to propositions and only the A path operator is used,

CTL* obtained from **TCTL*** with only trivial subscript intervals $I = [0, \infty)$, adding modal operator "next" to the syntax,

ACTL* restriction of **CTL***, where negation can only be applied to propositions and only the A path operator is used.

Let $\pi = s_0 s_0^{\delta_0} s_1 s_1^{\delta_1} \ldots$ be a path of the model M, s.t. $s_i \xrightarrow{\delta_i} s_i^{\delta_i}$ and $s_i^{\delta_i} \xrightarrow{\alpha_i} s_{i+1}$ for $i \geq 0$, where $s_i^{\delta} = (l_i, \eta_i, v_i + \delta)$ for $\delta \in \mathbb{R}_+$ and $s_i = s_i^0$, and let $\pi(i, \delta) = s_i^{\delta} s_i^{\delta_i} s_{i+1} s_{i+1}^{\delta_{i+1}} \ldots$ denote the suffix of π for $i \geq 0$ and $\delta \leq \delta_i$, such that $s_i^{\delta} \xrightarrow{\delta_i - \delta} s_i^{\delta_i}$. The semantics of **TCTL*** is as follows:[7]

$M, s \models p$	iff $p \in L(s)$
$M, s \models \mathsf{A}\psi$	iff $M, \pi \models \psi$, for all $\pi \in Paths(s)$
$M, s \models \mathsf{E}\psi$	iff $M, \pi \models \psi$, for some $\pi \in Paths(s)$
$M, \pi \models \varphi$	iff $M, \pi[0] \models \varphi$
$M, \pi \models \psi_1\, \mathrm{U}_I\, \psi_2$	iff $\exists i \geq 0. \exists \delta \leq \delta_i. \left((\sum_{j<i} \delta_j + \delta) \in I \right)$, and $M, \pi(i, \delta) \models \psi_2$, and $\forall \delta' < \delta. M, \pi(i, \delta') \models \psi_1$, and $\forall j < i. \forall \delta' \leq \delta_j. M, \pi(j, \delta') \models \psi_1$.

The model M satisfies the **TCTL*** (state) formula φ (denoted by $M \models \varphi$) iff $M, ini \models \varphi$.

3 SIMULATION FOR TMAS GRAPHS

We now propose a notion of *simulation* between timed MAS graphs, that is later used to establish correctness of our abstraction scheme.

Definition 3.1. Let $M_i = (St_i, ini_i, \longrightarrow_i, AP_i, L_i)$, $i = 1, 2$. A model M_2 *simulates* model M_1 over $AP \subseteq AP_1 \cap AP_2$ (denoted $M_1 \precsim_{AP} M_2$) if there exists a *simulation* relation $\mathcal{R} \subseteq St_1 \times St_2$ such that:

(i) $(ini_1, ini_2) \in \mathcal{R}$, and
(ii) for all $(s_1, s_2) \in \mathcal{R}$:
  (a) $L_1(s_1) \cap AP = L_2(s_2) \cap AP$, and
  (b) if $s_1 \longrightarrow_1 s_1'$, then $\exists s_2' \in St_2$ s.t. $s_2 \longrightarrow_2 s_2'$ and $(s_1', s_2') \in \mathcal{R}$.

If additionally $(s_1, s_2) \in \mathcal{R} \Rightarrow (v_1 = v_2)$, where $s_i = (l_i, \eta_i, v_i)$ for $i = 1, 2$, then $\mathcal{R}$ is called the *timed simulation* relation [11].

Theorem 3.2. *If there is timed simulation $\mathcal{R} \subseteq St_1 \times St_2$ over $AP \subseteq AP_1 \cap AP_2$, then for any formula $\varphi \in$ **TACTL*** over AP:*

$$M_2 \models \varphi \quad \textit{implies} \quad M_1 \models \varphi.$$

Proof (Sketch). The results showing that simulation preserves **ACTL** and **ACTL*** are well established [1, 51], this can be proven using structural induction on formula (cf. [4, 52]). Almost the same line of reasoning can be applied to timed simulation and **TACTL***. Here, we show that for a more interesting case, when φ is of the form $\mathsf{A}\psi_1\, \mathrm{U}_I\, \psi_2$; the remaining cases are shown analogously as in **ACTL***.

Let $\mathcal{R} \subseteq St_1 \times St_2$ be a timed simulation for (M_1, M_2), and $\pi = s_{i,0} s_{i,0}^{\delta_{i,0}} s_{i,1} s_{i,1}^{\delta_{i,1}} \ldots$ denote a path of the model M_i for $i = 1, 2$, such that $s_{i,j} \xrightarrow{\delta_{i,j}} s_{i,j}^{\delta_{i,j}}$ and $s_{i,j}^{\delta_{i,j}} \xrightarrow{\alpha_{i,j}} s_{i,j+1}$, where $s_{i,j} = (l_{i,j}, \eta_{i,j}, v_{i,j})$ and $s_{i,j}^{\delta} = (l_{i,j}, \eta_{i,j}, v_{i,j} + \delta)$, and let $\pi_i(j, \delta) = s_{i,j}^{\delta} s_{i,j}^{\delta_{i,j}} s_{i,j+1} s_{i,j+1}^{\delta_{i,j+1}} \ldots$ denote the suffix of π for $j \geq 0$ and $\delta \leq \delta_{i,j}$, s.t. $s_{i,j}^{\delta} \xrightarrow{\delta_{i,j} - \delta} s_{i,j}^{\delta_{i,j}}$.

Let $\pi_1 \in Paths_{M_1}(ini_1)$ be an arbitrarily chosen path. From Definition 3.1 we construct a matching to π_1 path $\pi_2 \in Paths_{M_2}(ini_2)$, s.t. $(s_{1,j}, s_{2,j}), (s_{1,j}^{\delta_{1,j}}, s_{2,j}^{\delta_{2,j}}) \in \mathcal{R}$ for all $j \geq 0$. Since $\mathcal{R}$ is a timed simulation, it follows that $v_{1,j} = v_{2,j}$ and $\delta_{1,j} = \delta_{2,j}$ for all $j \geq 0$. From $M_2, ini_2 \models \mathsf{A}\psi_1\, \mathrm{U}_I\, \psi_2$, we know that for $\pi_2 \in Paths_{M_2}(ini_2)$ the following holds: $\exists j \geq 0. \exists \delta \leq \delta_{2,j}. (\sum_{k<j} \delta_{2,k} + \delta) \in I$, and $M_2, \pi_2(j, \delta) \models \psi_2$, and $\forall \delta' < \delta. M_2, \pi_2(j, \delta') \models \psi_1$, and $\forall k < j. \forall \delta' < \delta_{2,k}. M_2, \pi_2(k, \delta') \models \psi_1$; consequently, for π_1 it follows that: $\delta \leq \delta_{1,j}$ and $(\sum_{k<j} \delta_{1,k} + \delta) \in I$, and by induction that $M_1, \pi_1(j, \delta) \models \psi_2$, and $\forall \delta' < \delta. M_1, \pi_1(j, \delta') \models \psi_1$, and $\forall k < j. \forall \delta' < \delta_{1,k}. M_1, \pi_1(k, \delta') \models \psi_1$. Since π_1 was chosen arbitrarily, the same reasoning can be applied to any path in $Paths_{M_1}(ini_1)$; hence we have $M_1, ini_1 \models \mathsf{A}\psi_1\, \mathrm{U}_I\, \psi_2$. □

Given a timed agent graph G, its time-insensitive variant $G_{\ominus}$ is a timed agent graph that is an almost identical copy of G except that the set of clocks $\mathcal{X}$ is set to $\varnothing$. Essentially, it also means that in $G_{\ominus}$, for any $l \in Loc$ an invariant function is such that $\mathcal{I}(l) = \top$, and for any $(l, g, \mathfrak{cc}, \varsigma, \alpha, X, l') \in E$, $\mathfrak{cc}$ and X are replaced with $\top$ and $\varnothing$ respectively.

For a TMAS graph $MG = \{\!|G^1, \ldots, G^{\mathcal{N}}|\!\}$, a time-insensitive variant $MG_{\ominus}$ is given through the time-insensitive variant of the agent graphs composing it, that is $MG_{\ominus} = \{\!|G^1_{\ominus}, \ldots, G^{\mathcal{N}}_{\ominus}|\!\}$

Lemma 3.3. *Any evaluation $\eta \in Eval_{\mathcal{V}}$ reachable at $l \in Loc$ for an agent graph G must also be reachable at l for $G_{\ominus}$.*

Proof. Follows directly from Definition 2.4. □

Corollary 3.4. *Given a pair of agent graphs G_1 and G_2 with their local domain functions d_1 and d_2, such that $d_i : Loc_i \mapsto \mathcal{P}(Eval_{\mathcal{V}_i})$ maps every location $l \in Loc_i$ to the set of its reachable evaluations, i.e.*

[7] The omitted clauses for the Boolean connectives are immediate.

Algorithm 1: Abstraction of TMAS graph MG wrt $\mathcal{F}$

in $G_{MG_{\ominus}}$ compute an over-approx. of local domain d for $Args_R\mathcal{F}$
foreach *timed agent graph* $G^i \in MG$ **do**
  compute an abstract timed agent graph $\mathcal{A}^{may}_{\mathcal{F}}(G^i)$ w.r.t. $d_i = \{l^i \mapsto \bigcup_{l \in Loc^1 \times \ldots \times \{l^i\} \times \ldots \times Loc^N} d(l)\}$
return $\mathcal{A}^{may}_{\mathcal{F}}(MG) = \{\!|\mathcal{A}^{may}_{\mathcal{F}}(G^1), \ldots, \mathcal{A}^{may}_{\mathcal{F}}(G^N)|\!\}$

Algorithm 2: An over-approx. of the local domain for $W \subseteq \mathcal{V}$

OverApproxLocalDomain($G = G_{MG}, W = Args_R(f)$)
  foreach $l \in Loc$ **do**
    $l.d := \varnothing$
    $l.p := \varnothing$
    $l.color := white$
  $l_0.d := \{\eta(W) \mid \eta \in [\![g_0]\!]\}$
  $Q := \varnothing$
  Enqueue(Q, l_0)
  while $Q \neq \varnothing$
    $l :=$ ExtractMax(Q)
    VisitLoc(l, W)
    if $l.color \neq black$ **then**
      foreach $l' \in Succ(l)$ **do** } enqueue immediate-neighbours
        $Q :=$ Enqueue(Q, l')
        $l'.p := l'.p \cup \{l\}$
      $l.color := black$
  return $\{l \mapsto \eta \mid l \in Loc, \eta \in Eval_{\mathcal{V}}, \eta(W) \in d.l\}$
VisitLoc(l, W)
  $\kappa := l.d$ } process incoming edges
  foreach $l' \in l.p$, $l' \xrightarrow{g,\mathfrak{cc},-,\alpha,X} l$ **do**
    $l.d := l.d \cup$ ProcEdge($l', g, \mathfrak{cc}, -, \alpha, X, l, W$)
  $l.p = \varnothing$
  if $\kappa \neq l.d$ **then**
    $l.color := grey$
  $\lambda := l.d$ } process self-loops
  foreach $l \xrightarrow{g,\mathfrak{cc},-,\alpha,X} l$ **do**
    $l.d := l.d \cup$ ProcEdge($l, g, \mathfrak{cc}, -, \alpha, X, l, W$)
  if $\lambda \neq l.d$ **then**
    $l.color := grey$
    go to 23
ProcEdge($l, g, \mathfrak{cc}, -, \alpha, X, l', W$)
  $H_{pre} := \{\eta \in [\![g]\!] \mid \eta(W) \in l.d\}$
  $H_{post} := \{Effect(\alpha, \eta) \mid \eta \in H_{pre}\}$
  return $\{\eta(W) \mid \eta \in H_{post}\}$

$d_i(l) = \{\eta \in Eval_{\mathcal{V}_i} \mid (l, \eta, v) \in Reach(\mathcal{M}(G_i))\}$*, for* $i = 1, 2$*, in case* $G_2 = G_{1\ominus}$ *is a time-insensitive variant of* G_1*, then its local domain* d_2 *is an over-approximation of the local domain* d_1*, that is* $d_1(l) \subseteq d_2(l)$ *for all* $l \in Loc_1 = Loc_2$*.*

4 VARIABLE ABSTRACTION FOR TIMED MAS

This section presents the abstraction method that is defined for the modular representation as TMAS graph and intended to reduce the state space of the induced abstract model.

Algorithm 3: Abstraction procedure

ComputeAbstraction(G, d, f, Sc)
  $Z_0 := (f([\![g_0]\!]|_W))(Z)$
  $g_0 := g_0 \wedge (Z{=}Z_0)$
  $\eta_0 \in [\![g_0]\!]$
  $E_a := \varnothing$
  foreach $e := l \xrightarrow{g,\mathfrak{cc},\varsigma,\alpha,X} l'$ **do**
    if $\{l, l'\} \cap Sc = \varnothing$ **then** } out-of-scope edge
      $E_a := E_a \cup \{e\}$
    else
      foreach $\eta \in d(l)$ **do**
        $\alpha' := \alpha$
        if $l \in Sc$ **then**
          $\alpha' := (W := \eta(W)).\alpha'$
          $\alpha' := (Z := (\eta_0(Z))).\alpha'$
        if $l' \in Sc$ **then**
          $\alpha' := \alpha'.(Z := (f(\eta|_W))(Z))$
          $\alpha' := \alpha'.(W := \eta_0(W))$
        $g' := g[W = \eta(W)]$
        $E_a := E_a \cup \{(l, g', \mathfrak{cc}, \varsigma, \alpha', X, l')\}$
  $E := E_a$
  $\mathcal{V} := \mathcal{V} \cup Z$
  return G

4.1 Definition

As in [39], the abstraction process is composed of two subroutines: computation of the local domain approximation followed by abstract model generation. [8] The intuition behind this is informally presented in Algorithm 1.

Formally, the abstraction $\mathcal{A}^{may}_{\mathcal{F}}$ for TMAS graph MG is specified by the set of pairs $\mathcal{F} = \{(f_1, Sc_1), \ldots, (f_{n_{\mathcal{F}}}, Sc_{n_{\mathcal{F}}})\}$, where $f_i : Eval_{W_i} \mapsto Eval_{Z_i}$ is an abstract mapping function, such that $W_i \subseteq \mathcal{V}$, $Z_i \cap \mathcal{V} = \varnothing$ and $i \neq j \Rightarrow Z_i \cap Z_j = \varnothing$, and $Sc_i \subseteq Loc$ is the effective scope, for any $1 \leq i, j \leq n_{\mathcal{F}}$. Intuitively, the scope enables applying a finer-grain abstractions, only for the certain fragment of the system. We denote the domain and range of mapping function f_i by $Args_R(f_i) = W_i$ and $Args_N(f_i) = Z_i$. For simplicity, in the sequel we restrict the presentation to the case of singleton $\mathcal{F} = \{(f, Sc)\}$ — the changes needed for the general case are merely technical and shall become apparent afterwards.

Intuitively, the abstraction obtains significant improvements under the following conditions: the verified property is *independent* from the variables being removed (so that the abstraction is conclusive), and the removed variables are largely *independent* from those being kept (so that it significantly reduces the state space). The *independence* is of semantic nature, and we see no easy way to automatically select such variables. At this stage, it seems best to follow a domain expert advice.

As follows from the discussion in the extended version of [39], lower-approximation of local domain is usually of little use in practice, as it can only map a location with a singleton or an empty set. Therefore, here we focus only on the may-abstraction procedure

[8] Essentially, the former subroutine might be skipped, when the required approximation is provided by the user.

for the TMAS graph that is based on the over-approximation of local domain.

Local domain approximation. The `OverApproxLocalDomain` from Algorithm 2 on the input takes a timed agent G (usually being a combined TMAS graph) and a subset of its variables $W \subseteq \mathcal{V}$, and then traverses the locations of G in a priority-BFS manner computing for each location the set of its possibly reachable evaluations of W. With each visit to a location, an algorithm attempts to refine the approximation, until stability (in terms of set-inclusion) is obtained. Starting from the initial one, every location must be visited at least once, and shall be re-visited again whenever any of its predecessors gets their approximation updated. As the number of locations and edges, and cardinality of the variables domains are all finite, the procedure is guaranteed to terminate.

Let $d_{max} := \max\{|domain(v)| \mid v \in W\}$, $k = |\mathcal{V}|$, $r = |W|$, $n = |Loc|$, $m = |E|$. The initialization loop on lines 1–4 takes $O(n)$ and line 5 $O(r)$. With generic heap implementation of priority queue, lines 6 and 7 run in $\Theta(\log n)$ and $\Theta(1)$ respectively. The loop on lines 8–15 repeats at most d^r_{max} times for each of n locations until stable approximation is obtained. Line 9 is in $O(\log n)$, lines 11–15 in $O(n^2)$. Upon visiting all n locations d^r_{max} times, `VisitLoc` calls to `ProcEdge` at most md^r_{max} times. Hence, given the set-union is in $O(d^r_{max})$ and *Effect* can be computed in $O(1)$, the running time of the whole `OverApproxLocalDomain` is $O(md^r_{max}(k + d^r_{max}) + n^3 d^r_{max})$. It needs $O(n)$ space for priority queue, $O(md^r_{max})$ for auxiliary look-up tables (e.g., for the pre-computation of $[\![g]\!]$) and $O(nd^r_{max} + n^2)$ for storing locations with their attributes. Hence, `OverApproxLocalDomain` is in $O(d^r_{max}(n + m) + n^2)$ space.

Abstraction generation. The abstraction computation procedure is described in Algorithm 3. In the main loop (lines 5–18) the edges of timed agent graph G are transformed according to $\mathcal{F} = \{(f, Sc)\}$ as follows:

- the edges entering or within Sc have their actions appended with (1) update of the target variable Z and (2) update which sets the values of the source variables W to their defaults (resetting those),
- the edges leaving or within Sc have actions prepended with (1) update of source variables W (a temporarily one to be assumed for the original action) and (2) update which resets the values of the target variable Z.

Note that due to introduction of a scope, the variables in X cannot be genuinely removed for a proper subset of locations — instead, their evaluation are fixed to some constant value within the states, where the corresponding location label falls under the scope.

The lines 1–4 run in $O(r)$, the outer loop on lines 5–17 repeats exactly m times, the "if-else" condition check involves set operation and runs in $O(n)$. For the worst-case analysis we shall assume that it always proceeds with "else" block. The inner loop on the lines 8–17 repeats at most d^r_{max} times. Assuming that operations on lines 11–12, 14–15 are in $O(1)$ time,[9] we can conclude that `ComputeAbstraction` runs in $O(r + m(n + md^r_{max}))$. It requires at most $O(md^r_{max})$ for storing new edges, and $O(n)$ for $\mathcal{F}$[10]. Hence, space complexity of `ComputeAbstraction` is in $O(md^r_{max} + n)$.

4.2 Correctness

THEOREM 4.1. *Let* $M_1 = \mathcal{M}(MG)$, $M_2 = \mathcal{M}(\mathcal{A}^{may}_{\mathcal{F}}(MG))$, *where* $Args_R(\mathcal{F}) = W \subseteq \mathcal{V}$, $Args_N(\mathcal{F}) = Z$, $Z \cap \mathcal{V} = \varnothing$. *Then, for any* $V \subseteq (\mathcal{V} \setminus W)$ *and* $AP \subseteq Cond_V \cup Loc$, *the relation* $\mathcal{R} \subseteq St_1 \times St_2$, *defined by* $(l_1, \eta_1, v_1)\mathcal{R}(l_2, \eta_2, v_2)$ *iff* $l_1 = l_2$, $v_1 = v_2$ *and either* $(l_1 \in Sc) \wedge (\eta_1(V) = \eta_2(V))$, *or* $\eta_1(\mathcal{V}) = \eta_2(\mathcal{V})$, *is the timed simulation over AP between* M_1 *and* M_2.

PROOF. First, we show that condition (i) from Definition 3.1 holds. By construction of $\mathcal{A}^{may}_{\mathcal{F}}(MG)$ from Algorithm 3, an abstract initial condition is of the form $g_0 \wedge g_{Z_0}$ (cf. Line 2), where g_{Z_0} determines for Z its initial value $Z_0 := f([\![g_0]\!]|_W)(Z)$, that is $g_{Z_0} \equiv (Z{=}Z_0)$ (cf. Line 1). Let $ini_i = (l_0, \eta_i, v_i)$, where $\forall_{0 \le j \le n_X} v_i(x_j) = 0$ for $i = 1, 2$, and $\eta_1 \in [\![g_0]\!]$, $\eta_2 \in [\![g_0 \wedge g_{Z_0}]\!]$. Observe that $\eta_2 \in [\![g_0]\!]$ also holds, that is $\eta_1(\mathcal{V}) = \eta_2(\mathcal{V})$ (and therefore also $\eta_1(V) = \eta_2(V)$), and thus $(ini_1, ini_2) \in \mathcal{R}$.

Next, we show that condition (ii) from Definition 3.1 holds as well. Note that (ii-a) trivially holds due to the considered propositions in AP ranging over $V \subseteq (\mathcal{V} \setminus W)$ only, and the fact that $\mathcal{R}$ implies that related states agree on the evaluations for V. Therefore, for any $(s_1, s_2) \in \mathcal{R}$ we have that $L_1(s_1) \cap AP = L_2(s_1) \cap AP$, and so only (ii-b) remains to be shown.

Let $s_i, s'_i \in St_i$, $s_i = (l_i, \eta_i, v_i)$, $s'_i = (l'_i, \eta'_i, v'_i)$ for $i = 1, 2$, $(s_1, s_2) \in \mathcal{R}$ and $s_1 \longrightarrow_1 s'_1$. From Definition 2.4, this must be either a delay-transition or an action transition.

In the former case, $s_1 \xrightarrow{\delta}_1 s'_1$ for some $\delta \in \mathbb{R}_+$, s.t. (by Definition 2.4) $l_1 = l'_1$, $\eta_1 = \eta'_1$, $v'_1 = v_1 + \delta$, and $v_1, v_1 + \delta \in [\![\mathcal{I}_1(l_1)]\!]$. From $(s_1, s_2) \in \mathcal{R}$ we know that $l_1 = l_2$ and $v_1 = v_2$. From Algorithm 3 it follows that $\mathcal{I}_1 = \mathcal{I}_2$, implying that $v_2, v_2 + \delta \in [\![\mathcal{I}_2(l_1)]\!]$, so there must exist a (corresponding) delay-transition $s_2 \xrightarrow{\delta}_2 s'_2$, where $v'_2 = v_2 + \delta$, $l_2 = l'_2$, $\eta_2 = \eta'_2$. Hence, we have $(s_2, s'_2) \in \mathcal{R}$.

In the latter case, $s_1 \xrightarrow{\alpha_1}_1 s'_1$ for some $e_1 = l_1 \xhookrightarrow{g_1, cc_1, -, \alpha_1, X_1}_1 l'_1$, s.t. $\eta_1 \in [\![g_1]\!]$, $v_1 \in [\![cc_1]\!]$, $\eta'_1 = Effect(\alpha_1, \eta_1)$, $v'_1 \in [\![\mathcal{I}(l'_1)]\!]$ and $v'_1 = v_1[X_1 = 0]$. By Algorithm 3, each (concrete) labelled edge $e = (l, g, cc, \varsigma, \alpha, X, l') \in E$ (cf. Line 5) is associated with the set of matching (abstract) labelled edges $\{e_a(\eta(W)) \mid \eta \in d(l)\} \subseteq E_a$ (cf. Line 9) of the form $e_a(k) = (l, g[W = k], cc, \varsigma, \alpha^{post}(k) \circ \alpha \circ \alpha^{pre}(k), X, l')$ (cf. Line 18), such that:

- $\alpha^{pre}(k)$ corresponds to the consecutive assignment statements $Z := Z_0$ and $W := k$ if $l \in Sc$ (cf. Lines 11 to 13), and to τ ("do nothing" instruction) otherwise,
- $\alpha^{post}(k)$ corresponds to the consecutive assignment statements $Z := (f(\eta|_W))(Z)$ and $W := \eta_0(W)$ for $\eta_0 \in [\![g_0]\!]$, if $l' \in Sc$ (cf. Lines 14 to 16), and to τ otherwise.

Since $(s_1, s_2) \in \mathcal{R}$, $v_1 \in [\![cc]\!]$, and $v'_1 \in [\![\mathcal{I}_1(l'_1)]\!]$, it follows that $v_2 \in [\![cc]\!]$ and $v_2[X_1 = 0] \in [\![\mathcal{I}_2(l'_1)]\!]$. Furthermore, there must exist an action-transition $s_2 \xrightarrow{\alpha_2}_2 s'_2$ induced by a labelled edge $e_2 = e_a(\eta_1(W)) = (l_2, g_1[W = \eta_1(W)], cc, -, \alpha^{post}_1(\eta_1(W)) \circ \alpha_1 \circ \alpha^{pre}_1(\eta_1(W)), X_1, l'_2)$ matching to e_1, where $l_1 = l_2$, $l'_1 = l'_2$, (by Corollary 3.4 it must be that $\eta_1 \in d(l_1)$). Since $\eta_1 \in [\![g]\!]$ and $\eta_1(V) =$

[9] In general, having a complex or lengthy edge labels is considered a bad practice, which significantly degrades the model readability; realistically, the lengths of edge labels can be assumed to be relatively small.

[10] The latter would be $O(nn_{\mathcal{F}})$ when $\mathcal{F}$ is not a singleton.

V	C	Concrete			A1			A2			A3		
		#St	**M**	**t**	**#St**	**M**	**t**	**#St**	**M**	**t**	**#St**	**M**	**t**
1	1	1.1e+2	9	0	1.1e+2	9	0	3.9e+1	9	0	3.9e+1	9	0
1	2	1.4e+2	9	0	1.4e+2	9	0	4.9e+1	9	0	4.9e+1	9	0
1	3	1.7e+2	9	0	1.7e+2	9	0	5.9e+1	9	0	5.9e+1	9	0
2	1	5.5e+3	9	0	5.5e+3	9	0	7.4e+2	9	0	4.7e+2	9	0
2	2	9.0e+3	9	0	9.0e+3	9	0	1.2e+3	9	0	7.8e+2	9	0
2	3	1.3e+4	10	0	1.3e+4	10	0	1.7e+3	9	0	1.2e+3	10	0
3	1	2.7e+5	33	1	2.7e+5	34	1	1.4e+4	10	0	5.3e+3	10	0
3	2	5.8e+5	61	2	5.8e+5	61	2	2.7e+4	11	0	1.2e+4	11	0
3	3	1.0e+6	100	3	1.0e+6	100	3	4.7e+4	13	0	2.2e+4	12	0
4	1	1.3e+7	1 182	59	1.3e+7	1 182	59	2.5e+5	30	1	5.8e+4	15	0
4	2	3.6e+7	3 247	163	3.6e+7	3 248	162	6.1e+5	63	3	1.7e+5	26	2
4	3	7.9e+7	7 113	369	7.9e+7	7 113	369	1.3e+6	122	6	4.0e+5	45	5
5	1	MEMOUT			MEMOUT			4.4e+6	396	24	6.2e+5	64	5
5	2	MEMOUT			MEMOUT			1.4e+7	1 187	78	2.4e+6	225	31
5	3	MEMOUT			MEMOUT			3.5e+7	3 079	208	7.1e+6	621	118
6	1	MEMOUT			MEMOUT			7.7e+7	6 738	541	6.3e+6	554	67
6	2	MEMOUT			MEMOUT			3.0e+8	26 554	2 306	3.4e+7	3 002	523
6	3	MEMOUT			MEMOUT			MEMOUT			1.2e+8	10 528	2 542
7	1	MEMOUT			MEMOUT			MEMOUT			6.4e+7	5 408	821
7	2	MEMOUT			MEMOUT			MEMOUT			MEMOUT		

Table 1: Experimental results for model checking φ_1 (FAA) on concrete and abstract models with no re-voting

$\eta_2(V)$, it follows that $\eta_2 \in [\![g[W = \eta_1(W)]]\!]$. Moreover, given that $\mathit{Effect}(\alpha^{post}(\eta_1(W)) \circ \alpha \circ \alpha^{pre}(\eta_1(W)), \eta_2) = \eta_2'$, we have $\eta_2'|_{\mathcal{V}}[W = \eta_1'(W)] = \mathit{Effect}(\alpha, \eta_2|_{\mathcal{V}}[W = \eta_1(W)])$, or, in other words, $\eta_1'(V) = \eta_2'(V)$. Hence, we conclude that $(s_2, s_2') \in \mathcal{R}$. □

5 EXPERIMENTAL EVALUATION

In this section we report the series of experiments on a voting system case study.

5.1 Benchmark: Estonian Internet Voting

We extend the Continuous-time Asynchronous Multi-Agent System (CAMAS) model of the voting scenario from [3], which was inspired by the election procedure in Estonia [55]. In particular, we add a malicious agent (Coercer), extra variables for the Election Authority (voting frequency and tallying), extra locations and labelled transitions for the Voters (interaction with coercer and possible re-voting). Furthermore, we parameterize the system with the number of voters (`NV`), the number of candidates (`NC`), the Boolean specifying whether re-voting is allowed (`RV`), the type of coercer's behaviour (`CTYPE`) and punishment criteria (`OBEY,DISOBEY`).

We use Uppaal model checker [57] and verify various configurations of the system — determined by its parameter values — with regard to the exposure of voters to the coercion through forced abstention or forced participation [40].

The voting scenario is standard: each voter (V) first needs to register for the preferred voting modality: postal vote, e-vote over the internet or traditional paper vote at a polling station; if time constraints for the chosen modality are met, the election authority (EA) accepts the registration and immediately provides appropriate voting material to that voter (e.g., election package, e-voting credentials, address of the assigned election commission office). Upon receiving these materials, V proceeds with either casting the vote for selected candidate, casting an invalid vote (e.g., by crossing more than one candidate) or abstaining from voting; if time constraints for casting the vote over the given medium are met, then EA records the vote in the tally. Then, V can interact with the coercer (C) once, and either get punished for not complying with the instructions or not (and possibly get rewarded). Finally, when the election time is over, EA closes the vote and C punishes all the V who did not show how they voted beforehand.

As in [3], we assign each voting modality a specific time frame: 1–7 for postal vote, 6–9 for e-vote, 10–11 for paper vote, and close the election at 11 time units.

The global (shared) variables:

- `sh,prev`: auxiliary variables used to pass the current and previously cast value of `vote` from V to EA,

The Election Authority timed agent graph's local variables:

- `tally`: an array of size `NC+1` storing the number of votes cast per candidate,[11]
- `freq`: voting frequency as the number of voters who

The Voter timed agent graph's local variables:

- `mode`: (currently) chosen voting modality,
- `vote`: encoding of cast vote corresponding to the candidate (`1..NC`), invalid vote (`0`) or no previously registered participation (`-1`).
- `p`: Boolean variable whether V was punished,
- `np`: Boolean variable whether V was not punished,[12]

In our model we distinguish two types of coercers with deterministic punishment/reward condition based on the shown receipt (here, any proof of how/whether V voted). The `TYPE1` coercer will punish a voter only when the `DISOBEY` condition matches the shown receipt (or voter refused to show it), whereas `TYPE2` will always punish voter except when the `OBEY` condition matches the receipt.

In particular, the forced abstention attack (FAA) is captured by `TYPE2` coercer with `OBEY=-1` (where `-1` represents that voter did not cast her vote and thus was not counted towards voting frequency); similarly, the forced participation attack (FPA) is captured by `TYPE1` coercer with `DISOBEY=-1`.

The FAA and FPA properties can be represented as follows:

$$(\varphi_1) \qquad \mathrm{AG}\,(\, V.np = \top \quad \Rightarrow \quad V.voted = \mathtt{OBEY}\,)$$

$$(\varphi_2) \qquad \mathrm{AG}\,(\, V.np = \top \quad \Rightarrow \quad V.voted \neq \mathtt{DISOBEY}\,)$$

φ_1 says that in all executions, V can avoid punishment only by abstaining from the voting (as instructed by C). φ_2 says that in all executions, V must take part in the voting to avoid getting punished.

5.2 Experiments and Results

For the experiments, we used a modified version of the open-source tool EasyAbstract[13], which implements the algorithms from [39] for Uppaal, to automate generation of the abstract models for each configuration of the system. Furthermore, we used a coarser variant of the local domain over-approximation based on the agent graph (or its template), where all synchronisation labels are simply discarded. By doing so, we were able to further reduce the memory and time usage. Due to FAA and FPA properties relating to the similar subset of atomic propositions, we employed the same abstractions in both cases:

[11] A greater size was used for technical reasons and to improve the readability; note that in practice `tally[0]=0` is the global invariant.
[12] Indeed, the pair of variables `p` and `np` is almost dual, however having both allows to distinguish the (initial) case when C has not yet decided whether to punish V or not.
[13] https://tinyurl.com/EasyAbstract4Uppaal

V	C	Concrete			A1			A2			A3		
		#St	**M**	**t**	**#St**	**M**	**t**	**#St**	**M**	**t**	**#St**	**M**	**t**
1	1	1.1e+2	9	0	1.1e+2	9	0	4.1e+1	9	0	4.1e+1	9	0
1	2	1.5e+2	9	0	1.5e+2	9	0	5.3e+1	9	0	5.3e+1	9	0
1	3	1.9e+2	9	0	1.9e+2	9	0	6.5e+1	9	0	6.5e+1	9	0
2	1	6.1e+3	9	0	6.1e+3	9	0	8.2e+2	9	0	4.9e+2	9	0
2	2	1.1e+4	9	0	1.1e+4	9	0	1.4e+3	9	0	8.4e+2	10	0
2	3	1.7e+4	10	0	1.7e+4	10	0	2.1e+3	9	0	1.3e+3	10	0
3	1	3.3e+5	38	1	3.3e+5	38	1	1.6e+4	10	0	5.6e+3	10	0
3	2	7.5e+5	75	2	7.5e+5	75	2	3.5e+4	12	0	1.3e+4	11	0
3	3	1.4e+6	137	4	1.4e+6	137	4	6.4e+4	14	0	2.4e+4	12	0
4	1	1.7e+7	1 533	73	1.7e+7	1 534	72	3.1e+5	36	1	6.1e+4	15	0
4	2	5.2e+7	4 535	218	5.2e+7	4 535	217	8.6e+5	83	3	1.9e+5	27	2
4	3	1.2e+8	10 743	531	1.2e+8	10 743	528	1.9e+6	176	8	4.5e+5	49	6
5	1	MEMOUT			MEMOUT			5.8e+6	507	30	6.5e+5	67	6
5	2	MEMOUT			MEMOUT			2.1e+7	1 841	113	2.7e+6	243	34
5	3	MEMOUT			MEMOUT			5.9e+7	5 020	325	7.9e+6	687	131
6	1	MEMOUT			MEMOUT			1.1e+8	9 170	722	6.7e+6	583	71
6	2	MEMOUT			MEMOUT			MEMOUT			3.7e+7	3 253	570
6	3	MEMOUT			MEMOUT			MEMOUT			1.4e+8	12 190	2 843
7	1	MEMOUT			MEMOUT			MEMOUT			6.8e+7	5 964	874
7	2	MEMOUT			MEMOUT			MEMOUT			MEMOUT		

Table 2: Experimental results for model checking φ_2 (FPA) on concrete and abstract models with no re-voting

A1: removes variables `tally,freq` in Election Authority TAG;
A2: in addition to A1 removes variable `mode` in Voter TAG(s);
A3: in addition to A2 removes variables `voted,p,np` in all Voter TAGs except one.

We report experimental results in the Tables 1 to 3.[14] The first two columns indicate the configuration, that is the number of voters ("V") and candidates ("C"); next, the two groups of columns aggregate the details of verification of the forced abstention attack ("FAA") and forced participation attack ("FPA") against the concrete and abstract ("A2" and "A3") TMAS graphs, within each group, when property is satisfied, the column "#St" indicates the number of symbolic states (as defined by Uppaal), "M" the amount of RAM used (in MiB[15]), "t" the time spent by CPU (in sec, rounded to the nearest whole number).[16] When model checking φ_1 (FAA) on models with re-voting allows, the verifier was always returning a counter-example run within <1 sec time.

It is noteworthy that with the help of abstraction, we were able to almost double the number of agents in the configuration before running out of memory due to the state space explosion (from 3–4 to 6–7 voters), effectively reducing the use of memory and time resources by up to two orders of magnitude. Note also that the effectiveness of the method heavily depended on the choice of the variables to remove – for instance, removing variables `tally,freq` (abstraction A1) did not prove useful at all.

6 CONCLUSIONS

In this paper, we propose a new scheme for *agent-based may abstractions of timed MAS*. The work extends the recent abstraction method [38], which was defined only for the untimed case. We also "lift" the algorithms of [39] to operate on MAS graphs with clocks, time invariants, and timing guards.

[14]The verification was performed on the machine with AMD EPYC 7302P 16-Core 1.5 GHz CPU, 32 GB RAM, Ubuntu 22.04, running `verifyta` command-line utility from Uppaal v4.1.24 distribution. The source code of the models and auxiliary scripts for running verification can be found on: https://github.com/aamas2025submission.
[15]1 MiB = 2^{20} Bytes, for more details see [37].
[16]The time spent on computing the abstract TMAS graph was negligible (< 1s) in all cases considered and is thus not included in the table.

V	C	Concrete			A1			A2			A3		
		#St	**M**	**t**	**#St**	**M**	**t**	**#St**	**M**	**t**	**#St**	**M**	**t**
1	1	2.7e+02	9	0	2.7e+02	9	0	1.0e+02	9	0	1.0e+02	9	0
1	2	3.7e+02	9	0	3.7e+02	9	0	1.4e+02	9	0	1.4e+02	9	0
1	3	4.6e+02	9	0	4.6e+02	9	0	1.8e+02	9	0	1.8e+02	9	0
2	1	3.4e+04	11	0	3.4e+04	12	0	5.1e+03	9	0	1.9e+03	9	0
2	2	6.3e+04	14	0	6.3e+04	14	0	9.6e+03	10	0	3.6e+03	10	0
2	3	1.0e+05	17	0	1.0e+05	17	0	1.6e+04	10	0	5.9e+03	10	0
3	1	4.2e+06	355	16	4.2e+06	355	16	2.4e+05	29	1	3.2e+04	12	0
3	2	1.1e+07	922	44	1.1e+07	922	43	6.3e+05	62	2	8.6e+04	17	1
3	3	2.2e+07	1 877	98	2.2e+07	1 877	95	1.3e+06	119	6	1.8e+05	26	3
4	1	MEMOUT			MEMOUT			1.1e+07	936	54	5.2e+05	53	4
4	2	MEMOUT			MEMOUT			3.9e+07	3 356	217	1.9e+06	173	27
4	3	MEMOUT			MEMOUT			1.0e+08	8 640	631	5.3e+06	461	109
5	1	MEMOUT			MEMOUT			MEMOUT			8.1e+06	679	88
5	2	MEMOUT			MEMOUT			MEMOUT			4.3e+07	3 647	750
5	3	MEMOUT			MEMOUT			MEMOUT			1.5e+08	12 821	3 879
6	1	MEMOUT			MEMOUT			MEMOUT			1.2e+08	10 151	1 617
6	2	MEMOUT			MEMOUT			MEMOUT			MEMOUT		

Table 3: Experimental results for model checking φ_2 (FPA) on concrete and abstract models with re-voting allowed

Similarly to [38, 39], our abstractions transform the specification of the system at the level of timed agent graphs, without ever generating the global model. An experimental evaluation, based on a scalable model of Estonian elections, have shown a very promising pattern of results. In all cases, computation of the abstract representation by our implementation took negligible time. Moreover, it allowed the Uppaal model checker to verify instances with state spaces that are several orders of magnitude larger. The experiments showed also that the effectiveness of the method depends on the right selection of variables to be removed; ideally, they should be provided by a domain expert.

In the future, we plan to refine Algorithm 2 and Algorithm 3, so that the approximation of local domain is computed over the pairs of location and zone, where the zone is an abstraction class of clock valuations that satisfy the same set of clock constraints occurring within the model and the formula [52]. Analogously, for the abstract TMAS generation, the labelled edges would be assigned the clock constraints corresponding to the possible zones of the source-location pair. This way, we hope to obtain more refined abstract models. It remains to be seen if the approach will turn out computationally feasible, or lead to the generation of an enormous (though finite) number of regions.

ACKNOWLEDGMENTS

The work has been supported by NCBR Poland and FNR Luxembourg under the PolLux/FNR-CORE project SpaceVote (POLLUX-XI/14/SpaceVote/2023 and C22/IS/17232062/SpaceVote), the PHC Polonium project MoCcA (BPN/BFR/2023/1/00045), CNRS IRP "Le Trójkąt", and ANR-22-CE48-0012 project BISOUS. For the purpose of open access, and in fulfilment of the obligations arising from the grant agreement, the authors have applied a Creative Commons Attribution 4.0 International (CC BY 4.0) license to any Author Accepted Manuscript version arising from this submission.

REFERENCES

[1] Rajeev Alur, Costas Courcoubetis, and David Dill. 1993. Model-checking in dense real-time. *Information and computation* 104, 1 (1993), 2–34.

[2] Rajeev Alur, Thomas A Henzinger, Orna Kupferman, and Moshe Y Vardi. 1998. Alternating refinement relations. In *Proceedings of CONCUR (Lecture Notes in Computer Science, Vol. 1466)*. 163–178.

[3] Jaime Arias, Wojciech Jamroga, Wojciech Penczek, Laure Petrucci, and Teofil Sidoruk. 2023. Strategic (Timed) Computation Tree Logic. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, AAMAS*. ACM, 382–390. https://doi.org/10.5555/3545946.3598661

[4] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of model checking*. MIT press.

[5] Thomas Ball and Orna Kupferman. 2006. An Abstraction-Refinement Framework for Multi-Agent Systems. In *Proceedings of Logic in Computer Science (LICS)*. IEEE, 379–388. https://doi.org/10.1109/LICS.2006.10

[6] Francesco Belardinelli, Rodica Condurache, Catalin Dima, Wojciech Jamroga, and Michal Knapik. 2021. Bisimulations for verifying strategic abilities with an application to the ThreeBallot voting protocol. *Information and Computation* 276 (2021), 104552. https://doi.org/10.1016/j.ic.2020.104552

[7] Francesco Belardinelli, Panagiotis Kouvaros, and Alessio Lomuscio. 2017. Parameterised Verification of Data-aware Multi-Agent Systems. In *Proceedings of IJCAI*. ijcai.org, 98–104. https://doi.org/10.24963/ijcai.2017/15

[8] Francesco Belardinelli and Alessio Lomuscio. 2017. Agent-based Abstractions for Verifying Alternating-time Temporal Logic with Imperfect Information. In *Proceedings of AAMAS*. ACM, 1259–1267.

[9] Francesco Belardinelli, Alessio Lomuscio, and Vadim Malvone. 2019. An Abstraction-Based Method for Verifying Strategic Properties in Multi-Agent Systems with Imperfect Information. In *Proceedings of AAAI*. 6030–6037.

[10] Francesco Belardinelli, Alessio Lomuscio, and Fabio Patrizi. 2011. Verification of Deployed Artifact Systems via Data Abstraction. In *Proceedings of ICSOC (Lecture Notes in Computer Science, Vol. 7084)*. Springer, 142–156. https://doi.org/10.1007/978-3-642-25535-9_10

[11] Patricia Bouyer, Uli Fahrenberg, Kim Guldstrand Larsen, Nicolas Markey, Joël Ouaknine, and James Worrell. 2018. Model checking real-time systems. *Handbook of model checking* (2018), 1001–1046.

[12] Thomas Brihaye, François Laroussinie, Nicolas Markey, and Ghassan Oreiby. 2007. Timed Concurrent Game Structures. In *Proceedings of CONCUR*. 445–459. https://doi.org/10.1007/978-3-540-74407-8_30

[13] Nils Bulling, Jurgen Dix, and Wojciech Jamroga. 2010. Model Checking Logics of Strategic Ability: Complexity. In *Specification and Verification of Multi-Agent Systems*, M. Dastani, K. Hindriks, and J.-J. Meyer (Eds.). Springer, 125–159.

[14] Alessandro Cimatti, Edmund M. Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. 2002. NuSMV2: An Open-Source Tool for Symbolic Model Checking. In *Proceedings of Computer Aided Verification (CAV) (Lecture Notes in Computer Science, Vol. 2404)*. 359–364.

[15] Edmund M. Clarke and E. Allen Emerson. 1981. Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic. In *Proceedings of Logics of Programs Workshop (Lecture Notes in Computer Science, Vol. 131)*. 52–71.

[16] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. 2000. Counterexample-Guided Abstraction Refinement. In *Proceedings of CAV (Lecture Notes in Computer Science, Vol. 1855)*. Springer, 154–169. https://doi.org/10.1007/10722167_15

[17] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. 2003. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM* 50, 5 (2003), 752–794. https://doi.org/10.1145/876638.876643

[18] Edmund M. Clarke, Orna Grumberg, and David E. Long. 1994. Model Checking and Abstraction. *ACM Transactions on Programming Languages and Systems* 16, 5 (1994), 1512–1542.

[19] Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem (Eds.). 2018. *Handbook of Model Checking*. Springer. https://doi.org/10.1007/978-3-319-10575-8

[20] Mika Cohen, Mads Dam, Alessio Lomuscio, and Francesco Russo. 2009. Abstraction in model checking multi-agent systems. In *Proceedings of (AAMAS*. IFAAMAS, 945–952.

[21] Patrick Cousot and Radhia Cousot. 1977. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages*. 238–252. https://doi.org/10.1145/512950.512973

[22] Dennis Dams, Rob Gerth, and Orna Grumberg. 1997. Abstract Interpretation of Reactive Systems. *ACM Trans. Program. Lang. Syst.* 19, 2 (1997), 253–291. https://doi.org/10.1145/244795.244800

[23] Dennis Dams and Orna Grumberg. 2018. Abstraction and Abstraction Refinement. In *Handbook of Model Checking*. Springer, 385–419. https://doi.org/10.1007/978-3-319-10575-8_13

[24] Luca de Alfaro, Patrice Godefroid, and Radha Jagadeesan. 2004. Three-Valued Abstractions of Games: Uncertainty, but with Precision. In *Proceedings of Logic in Computer Science (LICS)*. IEEE Computer Society, 170–179.

[25] JW De Bakker, Jan A. Bergstra, Jan Willem Klop, and J-J Ch Meyer. 1984. Linear Time and Branching Time Semantics for Recursion with Merge. *Theor. Comput. Sci.* 34 (1984), 135–156. https://doi.org/10.1016/0304-3975(84)90114-2

[26] E. Allen Emerson. 1990. Temporal and Modal Logic. In *Handbook of Theoretical Computer Science*, J. van Leeuwen (Ed.). Vol. B. Elsevier, 995–1072.

[27] Constantin Enea and Catalin Dima. 2008. Abstractions of multi-agent systems. *International Transactions on Systems Science and Applications* 3, 4 (2008), 329–337.

[28] Rob Gerth, Ruurd Kuiper, Doron Peled, and Wojciech Penczek. 1999. A Partial Order Approach to Branching Time Logic Model Checking. In *Proceedings of ISTCS*. IEEE, 130–139.

[29] Patrice Godefroid. 2014. May/Must Abstraction-Based Software Model Checking for Sound Verification and Falsification. In *Software Systems Safety*. Vol. 36. IOS Press, 1–16. https://doi.org/10.3233/978-1-61499-385-8-1

[30] Patrice Godefroid, Michael Huth, and Radha Jagadeesan. 2001. Abstraction-based model checking using modal transition systems. In *Proceedings of CONCUR (Lecture Notes in Computer Science, Vol. 2154)*. Springer, 426–440.

[31] Patrice Godefroid and Radha Jagadeesan. 2002. Automatic Abstraction Using Generalized Model Checking. In *Proceedings of Computer Aided Verification (CAV) (Lecture Notes in Computer Science, Vol. 2404)*. Springer, 137–150. https://doi.org/10.1007/3-540-45657-0_11

[32] Patrice Godefroid, Aditya V. Nori, Sriram K. Rajamani, and SaiDeep Tetali. 2010. Compositional may-must program analysis: unleashing the power of alternation. In *Proceedings of POPL*. ACM, 43–56. https://doi.org/10.1145/1706299.1706307

[33] Rong Gu, Peter G Jensen, Danny B Poulsen, Cristina Seceleanu, Eduard Enoiu, and Kristina Lundqvist. 2022. Verifiable strategy synthesis for multiple autonomous agents: a scalable approach. *International Journal on Software Tools for Technology Transfer* 24, 3 (2022), 395–414.

[34] Rong Gu, Peter G Jensen, Cristina Seceleanu, Eduard Enoiu, and Kristina Lundqvist. 2022. Correctness-guaranteed strategy synthesis and compression for multi-agent autonomous systems. *Science of Computer Programming* 224 (2022), 102894.

[35] Arie Gurfinkel, Ou Wei, and Marsha Chechik. 2006. Yasm: A Software Model-Checker for Verification and Refutation. In *Proceedings of CAV (Lecture Notes in Computer Science, Vol. 4144)*. Springer, 170–174. https://doi.org/10.1007/11817963_18

[36] Xiaowei Huang and Ron Van Der Meyden. 2014. Symbolic Model Checking Epistemic Strategy Logic. In *Proceedings of AAAI Conference on Artificial Intelligence*. 1426–1432.

[37] IEC 60027-2 2000. *Letter symbols to be used in electrical technology — Part 2: Telecommunications and electronics*. International Standard. International Electrotechnical Commission, Geneva, Switzerland.

[38] Wojciech Jamroga and Yan Kim. 2023. Practical Abstraction for Model Checking of Multi-Agent Systems. In *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning, KR*. 384–394. https://doi.org/10.24963/KR.2023/38

[39] Wojciech Jamroga and Yan Kim. 2023. Practical Model Reductions for Verification of Multi-Agent Systems. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI*. ijcai.org, 7135–7139. https://doi.org/10.24963/IJCAI.2023/834

[40] Wojciech Jamroga, Yan Kim, Peter B Roenne, and Peter YA Ryan. 2024. "You Shall not Abstain!" A Formal Study of Forced Participation. In *Proceeding of the 9th Workshop on Advances in Secure Electronic Voting, Voting*.

[41] Wojciech Jamroga, Wojciech Penczek, Teofil Sidoruk, Piotr Dembiński, and Antoni Mazurkiewicz. 2020. Towards Partial Order Reductions for Strategic Ability. *Journal of Artificial Intelligence Research* 68 (2020), 817–850. https://doi.org/10.1613/jair.1.11936

[42] Magdalena Kacprzak, Alessio Lomuscio, and Wojciech Penczek. 2004. Verification of Multiagent Systems via Unbounded Model Checking. In *Proceedings of AAMAS*. IEEE Computer Society, 638–645. https://doi.org/10.1109/AAMAS.2004.10086

[43] Panagiotis Kouvaros and Alessio Lomuscio. 2017. Parameterised Verification of Infinite State Multi-Agent Systems via Predicate Abstraction. In *Proceedings of AAAI*. 3013–3020.

[44] Alessio Lomuscio and Wojciech Penczek. 2007. Symbolic Model Checking for Temporal-Epistemic Logics. *SIGACT News* 38, 3 (2007), 77–99. https://doi.org/10.1145/1324215.1324231

[45] Alessio Lomuscio, Hongyang Qu, and Franco Raimondi. 2017. MCMAS: An Open-Source Model Checker for the Verification of Multi-Agent Systems. *International Journal on Software Tools for Technology Transfer* 19, 1 (2017), 9–30. https://doi.org/10.1007/s10009-015-0378-x

[46] Alessio Lomuscio, Hongyang Qu, and Francesco Russo. 2010. Automatic Data-Abstraction in Model Checking Multi-Agent Systems. In *Model Checking and Artificial Intelligence (Lecture Notes in Computer Science, Vol. 6572)*. Springer, 52–68. https://doi.org/10.1007/978-3-642-20674-0_4

[47] Kenneth L. McMillan. 1993. *Symbolic Model Checking: An Approach to the State Explosion Problem*. Kluwer Academic Publishers.

[48] Kenneth L. McMillan. 2002. Applying SAT Methods in Unbounded Symbolic Model Checking. In *Proceedings of Computer Aided Verification (CAV) (Lecture*

Notes in Computer Science, Vol. 2404). 250–264.
[49] Doron A. Peled. 1993. All from One, One for All: on Model Checking Using Representatives. In *Proceedings of CAV (Lecture Notes in Computer Science, Vol. 697)*, Costas Courcoubetis (Ed.). Springer, 409–423. https://doi.org/10.1007/3-540-56922-7_34
[50] Wojciech Penczek and Alessio Lomuscio. 2003. Verifying Epistemic Properties of Multi-Agent Systems via Bounded Model Checking. In *Proceedings of AAMAS* (Melbourne, Australia). ACM Press, New York, NY, USA, 209–216.
[51] Wojciech Penczek and Agata Półrola. 2001. Abstractions and partial order reductions for checking branching properties of time Petri nets. In *Applications and Theory of Petri Nets 2001: 22nd International Conference, ICATPN 2001 Newcastle upon Tyne, UK, June 25–29, 2001 Proceedings 22*. Springer, 323–342.
[52] Wojciech Penczek and Agata Półrola. 2006. *Advances in Verification of Time Petri Nets and Timed Automata: A Temporal Logic Approach*. Studies in Computational Intelligence, Vol. 20. Springer. https://doi.org/10.1007/978-3-540-32870-4
[53] Ph. Schnoebelen. 2003. The Complexity of Temporal Model Checking. In *Advances in Modal Logics, Proceedings of AiML 2002*. World Scientific.
[54] Sharon Shoham and Orna Grumberg. 2004. Monotonic Abstraction-Refinement for CTL. In *Proceedings of TACAS (Lecture Notes in Computer Science, Vol. 2988)*. Springer, 546–560. https://doi.org/10.1007/978-3-540-24730-2_40
[55] Drew Springall, Travis Finkenauer, Zakir Durumeric, Jason Kitcat, Harri Hursti, Margaret MacAlpine, and J Alex Halderman. 2014. Security analysis of the Estonian internet voting system. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 703–715.
[56] Stavros Tripakis and Sergio Yovine. 2001. Analysis of timed systems using time-abstracting bisimulations. *Formal Methods in System Design* 18 (2001), 25–68.
[57] Uppsala University and Aalborg University. 2002. *UPPAAL*. https://uppaal.org