Satisfiability Checking for (Strategic) Timed CTL Using IMITATOR

Wojciech Penczek[1][a], Laure Petrucci[2][b] and Teofil Sidoruk[1][c]
[1] *Institute of Computer Science, Polish Academy of Sciences, Poland*
[2] *LIPN, CNRS UMR 7030, Université Sorbonne Paris Nord, France*

Keywords: Temporal Logics, Timed Automata, Multi-Agent Systems, Strategic Ability, Parametric Verification, Satisfiability Checking, Model Synthesis.

Abstract: The satisfiability problem for Timed **CTL** (**TCTL**) is well known to be undecidable in general. Therefore, we propose a bounded approach, involving parametric encoding of all possible timed automata up to a given size (with certain restrictions). For this parametric timed automaton and an input formula, we synthesise parameter values using the IMITATOR model checker, and thus obtain a concrete model in which the formula is satisfied (if one exists within the bound). In other words, we define a partial algorithm for **TCTL** satisfiability checking. Moreover, we show how to represent memoryless strategies of agents with imperfect information via an auxiliary set of IMITATOR parameters, thereby applying our method also to Strategic **TCTL** (**STCTL**), the recently proposed extension of **TCTL** with the strategic modality. We evaluate practical feasibility on three benchmarks, including scalable instances.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

1 INTRODUCTION

Strategic Timed Computation Tree Logic (**STCTL**), recently introduced in (Arias et al., 2023), has been defined with both the discrete and continuous time representations, an important advantage over Timed Alternating-time Temporal Logic (**TATL**), which only uses the former (Laroussinie et al., 2006). Furthermore, **STCTL** offers the extra expressiveness of the strategic modality over Timed **CTL** (**TCTL**), while retaining the same computational complexity of model checking for memoryless strategies of agents with imperfect information (ir-strategies). The latter semantics of strategic ability is usually better suited for modeling real-world scenarios, since it seldom makes sense for agents to have full knowledge about the global state of a system (perfect information). **STCTL** is therefore of practical interest for verifying strategic abilities, for example, to analyze and formally prove desired properties of electronic voting protocols (Jamroga et al., 2022) or in security scenarios. The multi-agent representation of security scenarios introduces a new perspective compared to formalisms like attack-defense trees (Petrucci et al., 2019; Arias et al., 2020), enabling the study of opposing coalitions of intruders and defenders within a strategic context. Furthermore, **STCTL** is particularly well-suited for specifying and verifying timed systems where a discrete representation is insufficient, such as in automated control. A model checking algorithm for this logic was demonstrated for the first time in (Arias et al., 2023).

In this work, we aim to tackle the other notable decision problem regarding **STCTL**, namely satisfiability. The main obstacle here is the well-known result stating that the satisfiability problem is undecidable in general for **TCTL** (Alur et al., 1993a), and thus also for **STCTL**. Nonetheless, one may still consider satisfiability checking for a class of models whose size is limited by some constant, analogous to bounded model checking (BMC). Even then, solving the problem requires overcoming the challenges presented by continuous time semantics, which induces infinitely many possible timed automata of any given size.

Thus, the main focus of our approach is on presenting a finite, parametric encoding that allows for representing all timed automata with a fixed number of locations. We then leverage the parametric verification capabilities of the IMITATOR model checker (André, 2021). Since the tool supports **TCTL**, we begin with (bounded) satisfiability checking for this logic. Extending the method to **STCTL** involves adding an auxiliary set of parameters in the IMITA-

[a] https://orcid.org/0000-0001-6477-4863
[b] https://orcid.org/0000-0003-3154-5268
[c] https://orcid.org/0000-0002-4393-3447

Penczek, W., Petrucci, L. and Sidoruk, T.
Satisfiability Checking for (Strategic) Timed CTL Using IMITATOR.
DOI: 10.5220/0013257700003890
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 17th International Conference on Agents and Artificial Intelligence (ICAART 2025) - Volume 1*, pages 144-155
ISBN: 978-989-758-737-5; ISSN: 2184-433X
Proceedings Copyright © 2025 by SCITEPRESS – Science and Technology Publications, Lda.

TOR model, so as to restrict the choices of coalition agents, ensuring they form valid ir-strategies. In fact, we use the same technique as in (Arias et al., 2023) for **STCTL** model checking.

Notably, using a parametric verifier in this manner allows not only for obtaining an entire model satisfying some input property, but also for *partial synthesis* of one or more (often critical) components of a larger system whose remaining parts are already specified. Therefore, in our experimental evaluation, we include both full and partial synthesis of models. The latter option is particularly relevant, given the high computational cost of full satisfiability checking, and could be used in practice for a number of applications, e.g., in automated control or security systems (Dubinin et al., 2015), offering a new approach to efficiently obtain controller modules enforcing the desired behaviour in such scenarios. In fact, the proposed method guarantees that the optimal, i.e., smallest possible, component will be produced, which follows directly from our iterative, bounded approach to satisfiability checking.

Related Work. **STCTL** was proposed in (Arias et al., 2023), including a method for model checking **STCTL** formulae over asynchronous multi-agent systems (AMAS) (Jamroga et al., 2020). Relevant logical formalisms include especially Alternating-time Temporal Logic **ATL*** and its subset **ATL** (Alur et al., 2002), which extend linear temporal logic **LTL** with the strategic modality (in case of **ATL** it has to be immediately followed by a single temporal operator), and over the last two decades have become the most popular logics for reasoning about the strategic abilities of agents. **ATL*** model checking algorithms have been proposed for memoryless and perfect recall strategies (Alur et al., 2002) of agents with imperfect and perfect information (Jamroga, 2015) about the state of the system (except for the combination of imperfect information and perfect recall, where the problem is undecidable (Dima and Tiplea, 2011)). They have been applied for the verification of synchronous (Lomuscio and Raimondi, 2006) as well as asynchronous models (Jamroga et al., 2022); in the latter case, efficient reduction techniques have been adapted from **LTL** to mitigate state explosion (Jamroga et al., 2020). **TATL** (Laroussinie et al., 2006) extends **ATL** with time representation; several variants of **TATL** have been investigated, including a counting strategy semantics (see comparison in (Knapik et al., 2019)), but only in conjunction with discrete time.

On the other hand, the satisfiability problem has not been investigated nearly as broadly for the aforementioned logics, due to its significantly higher complexity than model checking (e.g., **EXPTIME** vs. **PTIME** for **ATL** with perfect information). One approach involves tableau-based procedures, successfully used for both **ATL** (Goranko and Shkatov, 2009) and **ATL*** (David, 2015). The other leverages Boolean monotonic theories (SMMT) solvers, originally proposed for **CTL** (Klenze et al., 2016), later extended to **ATL** (Kacprzak et al., 2020; Niewiadomski et al., 2020), and even a subset of Strategy Logic (Kacprzak et al., 2021), a more expressive formalism originally proposed in (Belardinelli et al., 2019).

Timed extensions of these temporal and strategic logics present a further challenge to satisfiability checking. In particular, adopting the continuous semantics of time typically means the problem becomes undecidable, such as for the real-time extensions of classical linear and branching temporal logics: Metric Temporal Logic **MTL** (Bouyer, 2009; Alur and Henzinger, 1993) and **TCTL** (Alur et al., 1993a). This motivates the pursuit for bounded approaches, such as the one presented in this paper. In particular, (Kacprzak et al., 2023) proposes such a method for Strategic **MTL** (**SMTL**), obtaining partial model synthesis via SMT-solvers combined with parametric model checking,

Finally, the recent paper (Arias et al., 2024) considers **STCTL** model checking and partial model synthesis, making it the closest related work. The investigated approach is declarative, based on rewriting logic (Meseguer, 1992; Meseguer, 2012), and includes practical evaluation using the Maude verifier, obtaining promising results. However, both the partial specification and the synthesised component in (Arias et al., 2024) do not contain any timing constraints, effectively making it an instance of the problem for **SCTL**, the untimed subset of the logic. As such, the results are not directly comparable with those obtained by our method, which also considers timing constraints in synthesised automata and therefore is much more broadly applicable.

Contribution. The novel contribution of the paper can be summarised as follows.

- We propose a partial algorithm for **STCTL** satisfiability checking (a problem undecidable in general), applicable to the fragment of the logic supported in the latest version of the verifier IMITATOR, i.e., non-nested formulas of the form $\langle\langle A\rangle\rangle\exists\mathtt{F}$, $\langle\langle A\rangle\rangle\forall\mathtt{F}$, and $\langle\langle A\rangle\rangle\forall\mathtt{G}\,\neg$. Since we allow for equalities in clock constraints (La Torre and Napoli, 2000), the problem remains undecidable also for this subset of **STCTL**, motivating our bounded approach.
- We experimentally evaluate our approach using several scalable benchmarks, including scenarios that involve both full and partial synthesis of timed automata networks.

Outline. The paper is structured as follows. Section 2 recalls the theoretical background of **STCTL** and the multi-agent formalism used. Then, we present our method in Section 3, first for **TCTL**, and extend it to **STCTL**. To demonstrate its practical feasibility, we generate scalable test instances and conduct experiments with IMITATOR, summarised in Section 4. Section 5 concludes the paper.

2 PRELIMINARIES

In this section, we recall from (Arias et al., 2023) the asynchronous multi-agent formalism with continuous (dense) time, and the logical framework for reasoning about strategic abilities of agents in such models.

2.1 Continuous-Time AMAS

Asynchronous Multi-Agent Systems (AMAS (Jamroga et al., 2020)) are similar to networks of automata that synchronise on shared actions, and interleave local transitions to execute asynchronously (Fagin et al., 1995; Lomuscio et al., 2010). However, to deal with agent coalitions, automata semantics must resort to algorithms and additional attributes. In contrast, by linking protocols to agents, AMAS are a natural compositional formalism to analyse multi-agent systems. Augmenting AMAS with continuous time involves some of the standard notions of *timed systems* (Alur and Dill, 1990). By $\mathbb{N}$ and $\mathbb{R}_{0+}$, we denote the sets of natural numbers and non-negative reals, respectively.

Definition 1 (Clocks)**.** *We denote a finite set of* clocks*, with a fixed ordering assumed for simplicity, by* $\mathcal{X} = \{x_1, \ldots, x_{|\mathcal{X}|}\}$*, where* $x_i \in \mathbb{R}_{0+}$*. A* clock valuation *on* $\mathcal{X}$ *is a* $|\mathcal{X}|$*-tuple* v*. We denote:*

- *by* $v(x_i)$ *or* $v(i)$*, the value of clock* x_i *in* v*;*
- *by* $v' = v + \delta$*, the increment of all clocks* $x \in \mathcal{X}$ *by* $\delta \in \mathbb{R}_{0+}$*;*
- *by* $v' = v[X := 0]$*, the reset of a subset of clocks* $X \subseteq \mathcal{X}$*, such that* $v'(x) = 0$ *for all* $x \in X$*, and* $v'(x) = v(x)$ *for all* $x \in \mathcal{X} \setminus X$*.*

Definition 2 (Clock constraints)**.** *The set* $\mathcal{C}_{\mathcal{X}}$ *collects all* clock constraints *over* $\mathcal{X}$*, defined by the grammar:* $\mathfrak{cc} := true \mid x_i \sim c \mid x_i - x_j \sim c \mid \mathfrak{cc} \wedge \mathfrak{cc}$*, where* $x_i, x_j \in \mathcal{X}$*,* $c \in \mathbb{N}$*, and* $\sim \in \{\leq, <, =, >, \geq\}$*. For* $\mathfrak{cc} \in \mathcal{C}_{\mathcal{X}}$*, the satisfaction relation* $\models$ *is inductively defined as:*

$v \models true$,
$v \models (x_i \sim c)$ *iff* $v(x_i) \sim c$,
$v \models (x_i - x_j \sim c)$ *iff* $v(x_i) - v(x_j) \sim c$*, and*
$v \models (\mathfrak{cc} \wedge \mathfrak{cc}')$ *iff* $v \models \mathfrak{cc}$ *and* $v \models \mathfrak{cc}'$.

$[\![\mathfrak{cc}]\!]$ *denotes the set of all valuations satisfying* $\mathfrak{cc}$.

In *continuous-time AMAS* (CAMAS), all agents have an associated set of clocks and a local invariant function defining a clock constraint for each local state. Clocks evolve at the same rate (across agents), thus allowing for delays and instantaneous actions.

Definition 3 (CAMAS)**.** *A* continuous-time AMAS (CAMAS) *consists of* n *agents* $\mathcal{A} = \{1, \ldots, n\}$*, each associated with a 9-tuple* $AG_i = (L_i, \iota_i, Act_i, P_i, \mathcal{X}_i, I_i, T_i, PV_i, V_i)$ *including:*

- *a finite non-empty set of* local states $L_i = \{l_i^1, l_i^2, \ldots, l_i^{n_i}\}$*;*
- *an* initial local state $\iota_i \in L_i$*;*
- *a finite non-empty set of* local actions $Act_i = \{a_i^1, a_i^2, \ldots, a_i^{m_i}\}$*;*
- *a* local protocol $P_i : L_i \to 2^{Act_i} \setminus \{\emptyset\}$*;*
- *a set of* clocks $\mathcal{X}_i$*;*
- *a* local invariant $I_i \colon L_i \to \mathcal{C}_{\mathcal{X}_i}$*;*
- *a (partial)* local transition function $T_i : L_i \times Act_i \times \mathcal{C}_{\mathcal{X}_i} \times 2^{\mathcal{X}_i} \rightharpoonup L_i$ *such that* $T_i(l_i, a, \mathfrak{cc}, X) = l_i'$ *for some* $l_i' \in L_i$ *iff* $a \in P_i(l_i)$*,* $\mathfrak{cc} \in \mathcal{C}_{\mathcal{X}_i}$*, and* $X \subseteq \mathcal{X}_i$*;*
- *a finite non-empty set of* local propositions $PV_i = \{\mathsf{p}_i^1, \ldots, \mathsf{p}_i^{r_i}\}$*;*
- *a* local valuation function $V_i : L_i \to 2^{PV_i}$*.*

For a local transition $t := l \xrightarrow{a, \mathfrak{cc}, X} l'$, l and l' are the *source* and *target* states, a is the executed *action*, clock condition $\mathfrak{cc}$ is called a *guard*, and X is the set of clocks to be *reset*. By $Agent(a) = \{i \in \mathcal{A} | a \in Act_i\}$, we denote the set of agents that "own" action a. Note that T_i is defined on local actions only. This is also reflected in the formal definition of CAMAS models, or Interleaved Interpreted Systems (Lomuscio et al., 2010; Jamroga et al., 2020).

Definition 4 (Model)**.** *The* model *of a CAMAS is an 8-tuple* $M = (\mathcal{A}, S, \iota, Act, \mathcal{X}, I, T, V)$*, including:*

- *the set of* agents $\mathcal{A} = \{1, \ldots, n\}$*;*
- *the set of* global states $S = \prod_{i=1}^n L_i$*;*
- *the* initial global state $\iota = (\iota_1, \ldots, \iota_n) \in S$*;*
- *the set of* actions $Act = \bigcup_{i \in \mathcal{A}} Act_i$*;*
- *the set of* clocks $\mathcal{X} = \bigcup_{i \in \mathcal{A}} \mathcal{X}_i$*;*
- *the* invariant $I(s) = \bigwedge_{i \in \mathcal{A}} I_i(s^i)$*, where* $s^i \in L_i$ *denotes agent* i*'s local component of global state* s*;*
- *the* global transition function $T : S \times Act \times \mathcal{C}_{\mathcal{X}} \times \mathcal{X} \to S$*, such that* $T(s, a, \bigwedge_{i \in Agent(a)} \mathfrak{cc}_i, X) = s'$ *iff* $\forall i \in Agent(a)$*,* $T_i(s^i, a, \mathfrak{cc}_i, X_i) = s'^i$*, whereas* $\forall i \in \mathcal{A} \setminus Agent(a)$*,* $s^i = s'^i$*;*
- *the* valuation function $V \colon S \to 2^{PV}$*, where* $PV = \bigcup_{i=1}^n PV_i$*.*

The continuous (dense) semantics of time defines *concrete states* as tuples of global states and non-negative real clock valuations.

Definition 5 (ACTS)**.** *The* concrete model *of a CAMAS model $M = (\mathcal{A}, S, \iota, Act, \mathcal{X}, I, T, V)$ is given by its* Asynchronous Continuous Transition System *(ACTS): a 5-tuple $\mathcal{M} = (\mathcal{A}, \mathcal{CS}, q_\iota, \rightarrow_c, V_c)$, including:*

- *the set of* agents $\mathcal{A} = \{1, \ldots, n\}$*;*
- *the set of* concrete states $\mathcal{CS} = S \times \mathbb{R}_{0+}^{|\mathcal{X}|}$*;*
- *the* concrete initial state $q_\iota = (\iota, v) \in \mathcal{CS}$*, s.t.* $\forall x_i \in \mathcal{X}, v(x_i) = 0$*;*
- *the* transition relation $\rightarrow_c \subseteq \mathcal{CS} \times (Act \cup \mathbb{R}_{0+}) \times \mathcal{CS}$*, defined by time- and action- successors as follows:* $(s,v) \xrightarrow{\delta}_c (s, v+\delta)$ *for* $\delta \in \mathbb{R}_{0+}$ *and* $v, v+\delta \in [\![I(s)]\!]$*,* $(s,v) \xrightarrow{a}_c (s', v')$ *iff there are* $a \in Act$*,* $\mathfrak{cc} \in \mathcal{C}_X$*,* $X \subseteq \mathcal{X}$ *s.t.:* $s \xrightarrow{a, \mathfrak{cc}, X} s' \in T$*,* $v \in [\![\mathfrak{cc}]\!]$*,* $v \in [\![I(s)]\!]$*,* $v' = v[X := 0]$*,* $v' \in [\![I(s')]\!]$*;*
- *the* valuation function $V_c((s,v)) = V(s)$*.*

Intuitively, delays $\xrightarrow{\delta}_c$ increase the clock valuation(s) by a given δ but do not change the global state, while actions $\xrightarrow{a}_c$ move to a successor state, possibly resetting some clocks.

Definition 6 (Execution)**.** *An execution of ACTS $\mathcal{M} = (\mathcal{A}, \mathcal{CS}, q_\iota, \rightarrow_c, V_c)$ from concrete state $q_0 = (s_0, v_0)$ is a sequence $\rho = q_0, \delta_0, q'_0, a_0, q_1, \delta_1, q'_1, a_1, \ldots$, where $q_i = (s_i, v_i)$, $q'_i = (s_i, v_{i+1})$, such that for each $i \geq 0$, we have: $\delta_i \in \mathbb{R}_{0+}$, $a_i \in Act$, $q_i \xrightarrow{\delta_i}_c q'_i \xrightarrow{a_i}_c q_{i+1}$.*

Note that due to the infinite number of concrete states, in practice we use the symbolic semantics of (parametric) zone graphs, where *zones* are convex polyhedra corresponding to clock constraints (Penczek and Pólrola, 2006). Moreover, we parametrise CAMAS by replacing constants with variables in clock constraints, as standard in Parametric Timed Automata (Penczek and Pólrola, 2006; André et al., 2016; André, 2019).

Untimed AMAS are a special case of Def. 3, where $\mathcal{X}_i = \emptyset$ for all $i \in \mathcal{A}$. Since the set of clocks is empty, the concrete model contains only action transitions, and thus it is identical to the model itself.

2.2 Strategies and Outcomes

Strategies are conditional plans that dictate the choice of an agent in each possible situation, classified based on agents' *state information*: perfect (I) vs. imperfect (i), and their *recall of state history*: perfect (R) vs. imperfect (r) (Schobbens, 2004). We consider strategies of type ir.

Definition 7 ((Joint) ir-strategy)**.** *A* memoryless imperfect information (ir) strategy *for $i \in \mathcal{A}$ is a function $\sigma_i \colon L_i \rightarrow Act_i$ such that $\sigma_i(l) \in P_i(l)$ for each $l \in L_i$.* *A* joint strategy *σ_A of coalition $A \subseteq \mathcal{A}$ is a tuple of strategies, one for each agent $i \in A$.*

The outcome of a joint strategy σ_A collects all executions consistent with σ_A, i.e., such that the coalition A strictly follows the strategy, while opponents freely choose actions allowed by their protocols.

Definition 8 (Outcome)**.** *Let $\mathcal{M}$ be an ACTS, $A \subseteq \mathcal{A}$, and $\rho = q_0, \delta_0, q'_0, a_0, q_1, \delta_1, q'_1, a_1, \ldots$, where $q_j = (s_j, v_j)$, $q'_j = (s_j, v_{j+1})$, an execution of $\mathcal{M}$. The* outcome *of* ir*-strategy σ_A in state q_0 of $\mathcal{M}$, denoted by $out^{\mathrm{ir}}_{\mathcal{M}}(q_0, \sigma_A)$, collects all executions ρ such that for each $i \in A$ and $j \geq 0$ we have: if $i \in Agent(a_j) \cap A$ then $a_j = \sigma_i(s^i_j)$.*

Providing the semantics for **STCTL** requires interpreting formulas along executions ρ of ACTS. To that end, we recall from (Penczek and Pólrola, 2006) the notion of a path corresponding to ρ.

Definition 9 (Corresponding path)**.** *Let ρ be an execution of an ACTS. By the dense path corresponding to ρ, denoted by π_ρ, we mean a mapping from $\mathbb{R}_{0+}$ to a set of concrete states, given by $\pi_\rho(r) = (s_i, v_i + \delta)$, for $r = \Sigma_{j=0}^{i-1} \delta_j + \delta$, where $i \geq 0$ and $0 \leq \delta < \delta_i$.*

2.3 Syntax and Semantics of STCTL

Assume a countable set PV of atomic propositions, and a finite set $\mathcal{A}$ of agents. The syntax of **STCTL** (Arias et al., 2023) is defined as:

$$\varphi ::= \mathsf{p} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\!\langle A \rangle\!\rangle \gamma,$$
$$\gamma ::= \varphi \mid \neg\gamma \mid \gamma \wedge \gamma \mid \forall \mathtt{X} \gamma \mid \exists \mathtt{X} \gamma \mid \forall \gamma \mathtt{U}_I \gamma \mid \exists \gamma \mathtt{U}_I \gamma,$$

where $\mathsf{p} \in PV$, $A \subseteq \mathcal{A}$, and $I \subseteq \mathbb{R}_{0+}$ is an interval with bounds $[n, n']$, $[n, n')$, $(n, n']$, (n, n'), (n, ∞), or $[n, \infty)$, for $n, n' \in \mathbb{N}$. The operator $\langle\!\langle A \rangle\!\rangle$ states that coalition A has a strategy to enforce the temporal property that follows. $\forall$ ("for all paths") and $\exists$ ("there exists a path") are standard **CTL** path quantifiers. `U` ("strong until") and `X` ("next", untimed only) are standard temporal operators. `G` ("always"), `F` ("eventually"), `R` ("release"), and Boolean connectives can be derived as usual. **SCTL** is obtained by restricting the time intervals to $I = [0, \infty)$ and removing them from the **STCTL** syntax. **TCTL** is obtained by removing the strategic modality $\langle\!\langle A \rangle\!\rangle$ from the **STCTL** syntax.

Note that we use a continuous-time representation, so the next-step operator `X` is not applicable when real-valued clocks are involved. However, following (Arias et al., 2023), we keep the operator `X` in the syntax, which allows to conveniently define other logics (**SCTL**, **CTL**) as subsets of **STCTL**.

Definition 10 (**STCTL** Semantics)**.** *Let $\mathcal{M} = (\mathcal{A}, \mathcal{CS}, q_\iota, \rightarrow_c, V_c)$ be an ACTS, $q_0 = (s, v) = (s_0, v_0) \in \mathcal{CS}$ a concrete state, $A \subseteq \mathcal{A}$, φ, ψ be* **STCTL**

formulae, $\rho = q_0, \delta_0, q_0', a_0, \ldots$ *an execution of* $\mathcal{M}$ *where* $q_k = (s_k, v_k)$*,* $q_k' = (s_k, v_{k+1})$*, and* π_ρ *be the dense path corresponding to* ρ*. The* ir*-semantics of* **STCTL** *is given as:*

- $\mathcal{M}, (s,v) \models \mathsf{p}$ *iff* $\mathsf{p} \in V_c(s,v)$*,*
- $\mathcal{M}, (s,v) \models \neg\varphi$ *iff* $M, (s,v) \not\models \varphi$*,*
- $\mathcal{M}, (s,v) \models \varphi \wedge \psi$ *iff* $M, (s,v) \models \varphi$ *and* $M, (s,v) \models \psi$*,*
- $\mathcal{M}, (s,v) \models \langle\langle A \rangle\rangle\gamma$ *iff there exists a joint strategy* σ_A *such that we have* $\mathcal{M}, out^{\mathrm{ir}}_{\mathcal{M}}((s,v),\sigma_A) \models \gamma$*, where:*
  - $\mathcal{M}, out^{\mathrm{ir}}_{\mathcal{M}}((s,v),\sigma_A) \models \varphi$ *iff* $M, (s,v) \models \varphi$*,*
  - $\mathcal{M}, out^{\mathrm{ir}}_{\mathcal{M}}((s,v),\sigma_A) \models \forall \mathtt{X}\gamma$ *iff for each* $\rho \in out^{\mathrm{ir}}_{\mathcal{M}}((s,v),\sigma_A)$ *we have* $\mathcal{M}, (\rho, \sigma_A, \mathrm{ir}) \models \mathtt{X}\varphi$*,*
  - $\mathcal{M}, out^{\mathrm{ir}}_{\mathcal{M}}((s,v),\sigma_A) \models \forall\gamma_1 U_I \gamma_2$ *iff for each* $\rho \in out^{\mathrm{ir}}_{\mathcal{M}}((s,v),\sigma_A)$ *we have* $\mathcal{M}, (\rho, \sigma_A, \mathrm{ir}) \models \gamma_1 U_I \gamma_2$*,*
  - $\mathcal{M}, out^{\mathrm{ir}}_{\mathcal{M}}((s,v),\sigma_A) \models \exists\gamma_1 U_I \gamma_2$ *iff for some* $\rho \in out^{\mathrm{ir}}_{\mathcal{M}}((s,v),\sigma_A)$ *we have* $\mathcal{M}, (\rho, \sigma_A, \mathrm{ir}) \models \gamma_1 U_I \gamma_2$*, where:*
    - $\mathcal{M}, (\rho, \sigma_A, \mathrm{ir}) \models \mathtt{X}\gamma$ *iff* $\mathcal{M}, out^{\mathrm{ir}}_{\mathcal{M}}(s_1, \sigma_A) \models \gamma$ *(untimed only)*
    - $\mathcal{M}, (\rho, \sigma_A, \mathrm{ir}) \models \gamma_1 U_I \gamma_2$ *iff there is* $r \in I$*:* $\mathcal{M}, out^{\mathrm{ir}}_{\mathcal{M}}((\pi_\rho(r)), \sigma_A) \models \gamma_2$ *and for all* $0 \leq r' < r$*:* $\mathcal{M}, out^{\mathrm{ir}}_{\mathcal{M}}((\pi_\rho(r')), \sigma_A) \models \gamma_1$*,*

3 ALGORITHMS FOR SATISFIABILITY CHECKING

In this section, we present our approach to **TCTL** satisfiability checking and then extend it for Strategic **TCTL**.

3.1 Satisfiability Checking for TCTL

It has been known for over three decades that in general, **TCTL** satisfiability is not decidable (Alur et al., 1993a). As such, no algorithm solving this problem exists. However, what we can offer is a partial algorithm; one that is guaranteed to stop, but can only prove the satisfiability (and not the unsatisfiability) of a given **TCTL** formula. The idea is as follows. We use the parametric model checker IMITATOR (André, 2021), which supports a fragment of **TCTL**, to check the satisfiability of the same fragment of the logic.[1] In fact, we aim at solving the problem of *bounded* satisfiability as we are able to check whether there exists a timed automaton of size up to some $k \in N$ whose model satisfies our formula. Consider a **TCTL** formula φ of the form $\varphi = \exists\mathtt{F}\ldots$, $\varphi = \forall\mathtt{F}\ldots$, or $\varphi = \forall\mathtt{G}\neg\ldots$, which corresponds to the subset supported by the latest version of IMITATOR, formally defined by the following grammar.

[1]Note that the problem remains undecidable for the fragment supported by IMITATOR, unless equalities in timing constraints are omitted (La Torre and Napoli, 2000).

Definition 11 (Supported subset of **TCTL**)**.** *The subset of* **TCTL** *supported by IMITATOR is defined as:*

$$\gamma ::= \varphi \mid \neg\gamma \mid \gamma \wedge \gamma \mid \forall\mathtt{F}_I\varphi \mid \exists\mathtt{F}_I\varphi \mid \forall\mathtt{G}_I\varphi,$$
$$\varphi ::= \mathsf{p} \mid \neg\varphi \mid \varphi \wedge \varphi.$$

where $\mathsf{p} \in PV$ *and* $I \subseteq \mathbb{R}_{0+}$ *are defined as in Sec. 2.3.*

Parametric Timed Automata. Crucially, IMITATOR is a *parametric* model checker, that is, it allows for input model (i.e., networks of automata analogous to CAMAS) in which the constants in state invariants and transition guards have been replaced by *parameters*. The output, given a **TCTL** formula and a Parametric Timed Automaton (PTA), is then one or more sets of parameter valuations for which the formula is satisfied, or UNSAT if no such valuations exist. For our purpose, it is sufficient to obtain just one valuation, therefore we use the `#witness` keyword rather than `#synth` in the IMITATOR property specification. Note that compared to PTA as originally proposed in (Alur et al., 1993b), the input models of IMITATOR are augmented with several features (André, 2021). We refer the reader to the tool's website and user manual available at https://www.imitator.fr for more details on the supported PTAs and their formal definitions.

Bounded Satisfiability Checking. In order to prove the (bounded) satisfiability of formula φ, one needs to demonstrate the existence of a timed automaton of size k such that φ holds in its model (Laroussinie et al., 1995). While it is not possible to generate all such timed automata, (since there are infinitely many due to the infinite number of clock expressions in guards and invariants), it is possible to symbolically encode all of them using a finite number of *parametric* clock expressions. Note that our approach can only be compared to Bounded Model Checking (BMC) in the sense that both of them set an upper bound within which to tackle a very challenging problem. The way BMC is done is completely different, and typically involves encoding (possibly symbolic) of finite paths in the model to SAT or SMT, and then leveraging the corresponding solver to handle subsequent instances.

In the following subsection, we discuss this encoding and formally define the parametric timed automaton (PTA) that symbolically represents all possible timed automata with k locations. Then, in Sec. 3.7 we discuss an extension of this approach to the logic **STCTL**, i.e., **TCTL** augmented with the strategic operator $\langle\langle\rangle\rangle$.

3.2 Parameters and Restrictions

In this work, our main aim is to establish the feasibility of the proposed approach to **TCTL** (and **STCTL**) satisfiability checking. To that end, we establish certain restrictions in order to limit the blow-up in the size of the IMITATOR model in practical experiments. In particular, to avoid duplicating all locations, we assume all invariants to be of one particular form, and use one clock per automaton.

Furthermore, when dealing with a *network* of timed automata, the different combinations of private and shared actions between local components induce yet additional complexity. To mitigate the issue, rather than synthesise the entire network, we aim at obtaining a single parametric timed automaton (PTA), and assume the specifications of the remaining parts of the network are known in advance (including which of their transitions are intended to be shared with the newly generated component).

We now detail the constants and parameters used in the IMITATOR model and the constraints imposed on them. Then, we formally define the parametric timed automaton that encodes all possible timed automata meeting these constraints.

- We restrict the number of clocks $x \in \mathbb{R}_{0+}$ to one per automaton.[2] Note that this class of timed automata remains a relevant subset studied in the literature (Laroussinie et al., 2004; Chen and Lu, 2008); moreover, it may be possible to reduce the number of clocks if more are present (Daws and Yovine, 1996; Guha et al., 2014). Clock expressions are of the form $x \sim c$, where $x \in \mathcal{X}$, $c \in \mathbb{N}$ and $\sim \in \{\leq, <, =, >, \geq\}$, i.e., only non-diagonal constraints.
- We encode a single parametric timed automaton. In other words, we either assume this PTA to represent an AMAS *model* (cf. Def. 4), or to be part of an automata network (i.e., AMAS) whose other components are known and already specified.
- By $k \in \mathbb{N}$, we denote the (fixed) number of locations in the automaton. Without loss of generality, we arbitrarily designate one of them as initial.
- Each location l^i is associated with one invariant, assumed to be of the form $x \leq p^I_{l^i}$, i.e., without diagonal clock constraints, where $p^I_{l^i} \in \mathbb{N}$ is the *invariant parameter*.
- There are k^2 possible transitions, one for each pair of locations (l^i, l^j), including self-loops.
- Each transition t is associated with a guard of one of two forms: $p^G_t \leq x \leq p'^G_t$ or $p^G_t \leq x$, i.e., without diagonal clock constraints, where $p^G_t, p'^G_t \in \mathbb{N}$ are the *guard parameters*. We will sometimes write $p^G_{(l,l')}$, $p'^G_{(l,l')}$ instead of p^G_t, p'^G_t to denote the guard parameters associated with transition $t : l \to l'$.
- We assume that for any pair of locations l^i, l^j, there exists at most one transition $l^i \to l^j$, i.e., multiple variants wrt. guard forms and clock resets (see below) cannot coexist in one concrete model.
- Due to different combinations of guards and clock resets, each transition t, for any source and target states l^i, l^j, occurs in one of the following variants $var \in \{1, \ldots, 4\}$:
  1. Transition $l^i \to l^j$ with a guard of the form $p^G_t \leq x \leq p'^G_t$, resetting the clock x,
  2. Transition $l^i \to l^j$ with a guard of the form $p^G_t \leq x$, resetting the clock x,
  3. Transition $l^i \to l^j$ with a guard of the form $p^G_t \leq x \leq p'^G_t$ without a clock reset,
  4. Transition $l^i \to l^j$ with a guard of the form $p^G_t \leq x$ without a clock reset,

  Note that the possibility where no transition exists from l^i to l^j is also covered; in this case IMITATOR will synthesise some other value $var \notin \{1, \ldots, 4\}$.

The restrictions on the types of guards and invariants are relatively minor and can always be relaxed, albeit at the cost of increasing the encoding size. Limiting each automaton to a single clock is somewhat more restrictive, but this class of automata remains an interesting subset frequently studied in the literature. The restriction to encoding a single parametric timed automaton is strongly motivated by its greater manageability and its suitability for typical scenarios, such as synthesizing a single controller based on the specifications of other existing modules.

3.3 Encoding

In the following, we assume $x \in \mathcal{X}$ is a clock, and the sets of *guards*, *invariants*, and *parameters* are as follows:

$Guards = \{p^G_t \leq x \leq p'^G_t, p^G_t \leq x\}$,

$Invariants = \{p^I_l \leq x\}$,

$Params = Params^G \cup Params^I \cup Params^T = \{p^G_{(l^1,l^1)}, \ldots, p'^G_{(l^k,l^k)}\} \cup \{p^I_{l^1}, \ldots, p^I_{l^k}\} \cup \{p^T_{l^1}, \ldots, p^T_{l^k}\}$,

where $p^G_{(l^i,l^j)}, p'^G_{(l^i,l^j)}$, and $p^I_{l^i}, p^T_{l^i}$ are associated with transition $l^i \to l^j$ and location l^i, respectively, and all are integers.

[2] If needed, we may also use an auxiliary variable $t \in \mathbb{R}_{0+}$ to represent elapsed global time in the IMITATOR model, see Sec. 4.

These are, respectively, the parameters that appear in guards, invariants, and transition variants. In the latter case, they are evaluated in equations of the form $PExpr = \{p^T_l = var \mid p^T_l \in Params \wedge var \in \{1, \dots, 4\}\}$ in the transitions of $\mathcal{PTA}^{P_\varphi}_k$, see below.

Definition 12. *The parametric timed automaton $\mathcal{PTA}^{P_\varphi}_k$ for φ, with $k \in \mathbb{N}$ locations and parameters $P = Params$, is defined as $\mathcal{PTA}^{P_\varphi}_k = (L, \iota, Act, \mathcal{X}, I, T, PV, V)$, where:*

- $L = \{l^1, l^2, \dots, l^k\}$;
- $\iota = l^1$;
- $Act = \{a^{(1,1)}, \dots, a^{(1,k)}, \dots, a^{(k,1)}, \dots, a^{(k,k)}\}$;
- $\mathcal{X} = \{x\}$;
- $I : L \to Invariants$,
- $T \subseteq L \times Act \times Guards \times PExpr \times \mathcal{X} \times L$, *defined as:*
  $T = \bigcup_{l,l' \in L}(T_1(l,l') \cup \dots \cup T_4(l,l'))$, *where:*
  $T_1(l,l') = \{(l, a^{(l,l')}, (p^G_{(l,l')} \leq x \leq p'^G_{(l,l')}), (p^T_l = 1), \{x\}, l')\}$,
  $T_2(l,l') = \{(l, a^{(l,l')}, (p^G_{(l,l')} \leq x), (p^T_l = 2), \{x\}, l')\}$,
  $T_3(l,l') = \{(l, a^{(l,l')}, (p^G_{(l,l')} \leq x \leq p'^G_{(l,l')}), (p^T_l = 3), \emptyset, l')\}$,
  $T_4(l,l') = \{(l, a^{(l,l')}, (p^G_{(l,l')} \leq x), (p^T_l = 4), \emptyset, l')\}$.
- $PV = Prop(\varphi) = \{\mathsf{p}_1, \mathsf{p}_2, \dots, \mathsf{p}_k\}$;
- $\forall_{l \in L} V(l) \in 2^{PV}$, *where PV is a set of Boolean variables corresponding to locations.*

Intuitively, when executing action $a^{(l,l')}$ from location l the automaton makes a choice, by setting the corresponding transition parameter p^T_l to $var \in \{1, \dots, 4\}$, to fire one of four copies $T_1(l,l')$, $T_2(l,l')$, $T_3(l,l')$, $T_4(l,l')$.

3.4 Complexity and Correctness

We now analyse the computational complexity of the encoding and show that the parametric timed automaton $\mathcal{PTA}^{P_\varphi}_k$ defined above encodes all timed automata of size k (assuming the restrictions detailed in Sec. 3.2).

Lemma 1. *The encoding of parametric timed automaton $\mathcal{PTA}^{P_\varphi}_k$ is of size $O(k^2)$ and uses $O(k^2)$ variables and parameters.*

Proof. Clearly, encoding $\mathcal{PTA}^{P_\varphi}_k$ in IMITATOR first requires exactly k variables corresponding to its $k \in \mathbb{N}$ locations. Note that each location induces one invariant parameter, one transition choice parameter, and $2k$ guard parameters (one guard per transition, indexed by its source and target states). Therefore, the total number of variables/parameters is $k \cdot (2k + 2) = 2k^2 + 2k$, and thus in $O(k^2)$.

Since we first initialize $2k^2 + 2k$ variables (see above), and then define $4k^2$ transitions (four variants per transition per location), the size of the encoding is $6k^2 + 2k$, which is also in $O(k^2)$. $\square$

Theorem 1. *$\mathcal{PTA}^{P_\varphi}_k$ encodes all the timed automata with k locations within the restrictions outlined above in Sec. 3.2.*

Proof. First, note that $\mathcal{PTA}^{P_\varphi}_k$ always has k locations, and all possible transitions between them are accounted for: from each location to the $k - 1$ others plus a self loop (for a total of k^2 transitions). Furthermore, it is not possible for multiple variants of a single transition to occur simultaneously, as they are associated with the same parameter (only one value of which will be synthesised by IMITATOR). This is consistent with the established restrictions on the structure of $\mathcal{PTA}^{P_\varphi}_k$, and each variant corresponds to the allowed forms of guards, with or without resets. For a discussion of the parameter synthesis algorithm implemented in IMITATOR and its correctness, we refer the reader to (André et al., 2019). $\square$

3.5 Using IMITATOR

IMITATOR supports parametric model checking for a fragment of **TCTL** (see Def. 11), including non-nested formulas φ of the form $\exists\mathtt{F}\dots$, $\forall\mathtt{F}\dots$ or $\forall\mathtt{G}\neg\dots$. For an input formula φ it returns the set P_φ of parameters for which we have that $\mathcal{M}, q_\iota \models \varphi$, where $\mathcal{M}$ is the concrete model (ACTS) of $\mathcal{PTA}^{P_\varphi}_k$. If $P_\varphi = \emptyset$, then φ is not satisfiable within the bound k on the number of locations in the automaton. While IMITATOR supports Boolean combinations of predicates embedded in formulas, relevant combinators are not provided for formulas themselves, e.g., $\exists\mathtt{F}\,\phi_1 \wedge \exists\mathtt{F}\,\phi_2$. However, one can run the verifier separately for both disjuncts (resp. conjuncts) and take the union (resp. intersection) of the resulting sets of parameters:

- for an input formula $\varphi = \varphi_1 \vee \varphi_2$, IMITATOR is run for φ_1 and for φ_2, and $P_\varphi = P_{\varphi_1} \cup P_{\varphi_2}$,
- for an input formula $\varphi = \varphi_1 \wedge \varphi_2$, IMITATOR is run for φ_1 and for φ_2, and $P_\varphi = P_{\varphi_1} \cap P_{\varphi_2}$.

3.6 Complete Satisfiability Checking

The procedure for checking whether a **TCTL** formula φ is satisfiable is as follows. We initially start Alg. 1 for $k = 1$.

Clearly, the algorithm is always guaranteed to terminate, returning SAT if the input formula φ is satisfiable within the chosen bound, or UNSAT otherwise. Moreover, in the former case it ends with the smallest

```
1: Run IMITATOR for input automaton $\mathcal{PTA}_k^{P_\varphi}$
   and property $\varphi$.
2: if $P_\varphi \neq \varnothing$ then
3:     return SAT
4: else if $(P_\varphi = \varnothing$ and $k < bound)$ then
5:     return $\mathrm{SAT}_{\mathbf{TCTL}}(\varphi, k+1, bound)$
6: else
7:     return UNSAT
8: end if
```

Algorithm 1: $\mathrm{SAT}_{\mathbf{TCTL}}(\varphi, k, bound)$

number of locations $k \leq bound$ which allows for the formula satisfaction (with our additional constraints specified in Sec. 3.2).

Example 1. *Let propositions in $PV = \{\mathsf{eat}_1, \mathsf{think}_1\}$ denote that a philosopher is dining and thinking, respectively. Consider the property $\phi = \exists \mathrm{F}_{[0,60]} \mathsf{think}_1$. For this small example, our PTA network will consist of a single automaton with the number of locations bounded to 2. Note that due to the way IMITATOR handles local variables, eat_1 and think_1 are effectively tied to local states. We begin with a model with 2 states and all possible transitions between them, defined as in Def. 12. In particular, we use parameters to ensure only one variant of each transition is present, see Fig. 1. Note that to model the time interval $[0,60]$ subscribed to temporal operator $\mathtt{F}$, we add an extra clock representing global time in IMITATOR.*

3.7 Satisfiability Checking for $\mathbf{STCTL_{ir}}$

We now discuss the extension of the approach presented above for **TCTL** to formulas of **STCTL**. Formally, the supported subset of **STCTL** is defined as in Def. 11, except the strategic modality $\langle\langle A \rangle\rangle$ is added to each element of γ.

The intuition behind this extension is analogous to the approach used in (Arias et al., 2023) where agents' strategies were encoded as parameters to leverage IMITATOR's capabilities for **STCTL** model checking.

Essentially, the change consists in further restricting the transition choice parameters $p_{l^i}^T$ in each location l_i of the parametric timed automaton, but only for the coalition agents. For a given agent coalition A, we encode its strategies on top of the existing approach detailed in Sec. 3.1 using a parametric timed automaton $\mathcal{PTA}(A)_k^{P_\varphi}$, defined as follows:

Definition 13. *Let $A \subseteq \mathcal{A}$ be an agent coalition. The automaton $\mathcal{PTA}(A)_k^{P_\varphi}$ is defined as previously in Def. 12, except for each location l^i, the corresponding parametric expression is defined as: $PExpr = \{p_l^T = var \mid p_l^T \in Params \wedge var \in \{1, \ldots, 4, \ldots, 4k\}\}$, and it is now checked in T as follows:*

```
var x, time: clock;
    p_guard_11, q_guard_11, p_guard_12, q_guard_12,
    p_guard_21, q_guard_21, p_guard_22, q_guard_22,
    p_invariant1, p_invariant2, t1, t2: parameter;

automaton pta

loc eat1: invariant x <= p_invariant1 & time <= 60
(* four copies of a^(1,1) *)
    when p_guard_11 <= x & x <= q_guard_11
       & t1 = 1 do {x:=0} goto eat1;
    when p_guard_11 <= x
       & t1 = 2 do {x:=0} goto eat1;
    when p_guard_11 <= x & x <= q_guard_11
       & t1 = 3 goto eat1;
    when p_guard_11 <= x & t1 = 4 goto eat1;
(* four copies of a^(1,2) *)
    when p_guard_12 <= x & x <= q_guard_12
       & t1 = 1 do {x:=0} goto eat2;
    when p_guard_12 <= x
       & t1 = 2 do {x:=0} goto eat2;
    when p_guard_12 <= x & x <= q_guard_12
       & t1 = 3 goto eat2;
    when p_guard_12 <= x & t1 = 4 goto eat2;
(* violation upon reaching 60 global time *)
    when time > 60 goto violation;

loc eat2: invariant x <= p_invariant2 & time <= 60
(* location eat2 defined analogously *)
[...]

loc violation: invariant True (* auxiliary location *)
end

init := {
    discrete = loc[pta] := eat1;
    continuous = time >= 0 & x = 0
               & p_guard_11 >= 0 & q_guard_11 >= 0
               & p_guard_12 >= 0 & q_guard_12 >= 0
               & p_guard_21 >= 0 & q_guard_21 >= 0
               & p_guard_22 >= 0 & q_guard_22 >= 0
               & p_invariant1 >= 0 & p_invariant2 >= 0
               & t1 > 0 & t2 > 0;
        }
end


property := #witness EF(loc[pta] = loc2);
```

Figure 1: IMITATOR model and property specification for Example 1.

$$p_{l^i}^T = 4 \cdot (i-1) + 1 \; for \; T_1(l, l'),$$
$$p_{l^i}^T = 4 \cdot (i-1) + 2 \; for \; T_2(l, l'),$$
$$p_{l^i}^T = 4 \cdot (i-1) + 3 \; for \; T_3(l, l'),$$
$$p_{l^i}^T = 4 \cdot (i-1) + 4 \; for \; T_4(l, l').$$

This ensures that the same choice is always made not just between the three variants of a transition, but across all outgoing transitions from a state, thus producing a valid ir-strategy.

We note that conjunctions of the form $\langle\langle A \rangle\rangle \varphi_1 \wedge$

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

Table 1: Results for the Dining Philosophers benchmark.

# states	result	running time (seconds)
2	sat	0.01
5	sat	0.16
10	sat	11.4
15	sat	199
20	sat	1856
25	memout	

$\langle\langle B\rangle\rangle\varphi_2$ can be handled via overapproximation, i.e., by taking the intersection $A \cap B$ of all involved agent coalitions and checking the satisfiability of formula $\langle\langle A\cap B\rangle\rangle\varphi_1 \wedge \langle\langle A\cap B\rangle\rangle\varphi_2$. The same approach could also be applied for nested strategic operators, however as pointed out in Sec. 3.1, nested formulas are currently not supported by IMITATOR.

4 EXPERIMENTAL RESULTS

In this section, we discuss experimental results. To demonstrate the practical feasibility of our approach, we used three sets of scalable benchmarks, detailed below. The IMITATOR binaries were run via Windows Subsystem for Linux (WSL) on a Windows 10 PC with a 4.0 Ghz CPU and 64 GB RAM. As per the default WSL configuration, the running process was terminated if its memory usage exceeded half of the total available, indicated by *memout*.

4.1 Full Synthesis

The first benchmark involves full synthesis of the entire CAMAS.

Simple Dining Philosophers. This is a scaled version of the simple model introduced in Example 1, with the number of locations n in the synthesised PTA ranging from 2 to 25. The set of propositional variables is $PV = \{\mathsf{eat_1}, \mathsf{think_1}, \ldots, \mathsf{eat_n}, \mathsf{think_n}\}$. Each location i is associated with propositions $\mathsf{eat_i}$ and $\bigcup_{1..n} \mathsf{think_j}$, where $j \neq i$.

The formula $\phi_1 = \exists \mathrm{F}_{[0,60]} \mathsf{think_n}$ remains the same as in Example 1. Experimental results are summarised in Table 1.

Notice that IMITATOR's running time is about half an hour for $n = 20$ locations, showing that our approach can be applied to models whose size exceeds simple toy examples. However, the scalability remains somewhat limited, and there remain other issues, especially the fact we currently synthesise a single PTA.

Nonetheless, it bears reminding that **TCTL** and **STCTL** synthesis is undecidable in general, and so

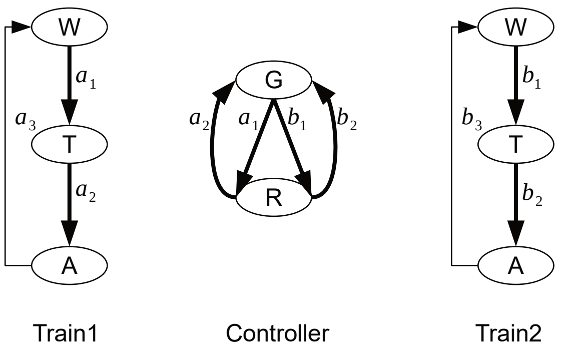

Figure 2: The AMAS for the TGC scenario. Transitions a_3 and b_3 are private; all others (highlighted in bold) may be shared.

is the bounded satisfiability problem for these logics, even assuming their restricted subsets supported by IMITATOR. As such, scaling to a larger number of locations was always expected to be challenging. Furthermore, obtaining small models satisfying a given property is typically preferable, and often the only feasible option given the complexity of the problem.

A potential alternative overcoming these pitfalls, which we investigate with the next test instances, involves synthesising a single, key component of a partially defined model, rather than the full one.

4.2 Partial Synthesis

In the remaining two scenarios, a single CAMAS component is obtained based on a partial model specification.

Train-Gate-Controller. Based on the classical Train-Gate-Controller (TGC) scenario (Alur et al., 1993b; Alur et al., 1998; Hoek and Wooldridge, 2002; Jamroga et al., 2020), this benchmark features two trains and a controller tasked with preventing both trains from entering a tunnel gate at the same time.

As opposed to the Simple Dining Philosophers example, we start with an automata network for TGC rather than its product (i.e., model). This allows for taking into account shared transitions between agents' local components. Furthermore, it enables partial synthesis where one or more automata are predefined. Here, we assume that the specifications for the trains are already given, while the controller satisfying the required property (two trains will never be together in the tunnel) is to be synthesised.

Unfortunately, there is a major drawback to synthesising local components in this manner. In particular, we have no convenient way of parameterising shared transitions in IMITATOR. Thus, depending on the number of non-private actions in the pre-

Table 2: Results for the Train-Gate-Controller benchmark.

# trains	result	running time (seconds)
2	sat	0.07
5	sat	0.53
10	sat	2.25
15	sat	5.16
20	sat	9.38
25	sat	14.5
50	sat	58.2
100	sat	284

defined local components of the trains, we need to enumerate as many choices for each transition variant in the synthesised automaton of the controller. As seen in Fig. 2, in this case there are four transitions a_1, a_2, b_1, b_2 of the trains that may or may not be shared with those of the controller. Therefore, not only does each of the latter's transitions come in 4 variants corresponding to different combinations of guards and resets, but each variant has 5 versions depending on whether it is shared with a_1, a_2, b_1, b_2, or private. This amounts to 20 copies of each transition, making the proposed approach rather unsuitable for all but the simplest specifications of the given components, unless of course they are assumed to use private transitions exclusively.

The set of propositional variables is $PV = \{\mathsf{away}_1, \mathsf{away}_2\}$, representing train 1 and train 2 having gone through the tunnel, respectively, and associated with the corresponding locations in their predefined local automata. The formula $\phi_2 = \langle\langle Controller \rangle\rangle \exists \mathsf{F}_{[0,60]} \mathsf{away}_1$ says that the controller has a strategy to let the first train through the tunnel within 60 time units. Since this is a **STCTL** property, we use transition choice parameters accordingly, as per Def. 13. That is, the range of t_1 and t_2 is $\{1, \ldots, 8\}$, rather than $\{1, \ldots, 4\}$ for non-strategic properties. The results for this benchmark, scaled with the number of trains, are presented in Table 2.

Indeed, it turns out that partial synthesis in TGC scales much better than the approach taken in the previous benchmark, even when the controller has a significant number of copied transitions that correspond to possible synchronisations with different trains. Clearly, the complexity induced by synchronisation with the predefined components is far outweighed by that stemming from increasing the bound on the synthesised automaton's size (the latter evident in the results for Dining Philosophers). This is good news, as it demonstrates that our method can be applied even in larger multi-agent models, provided that the component to be obtained remains relatively small (which, realistically, is always a requirement in today's state of the art, understandably so given the

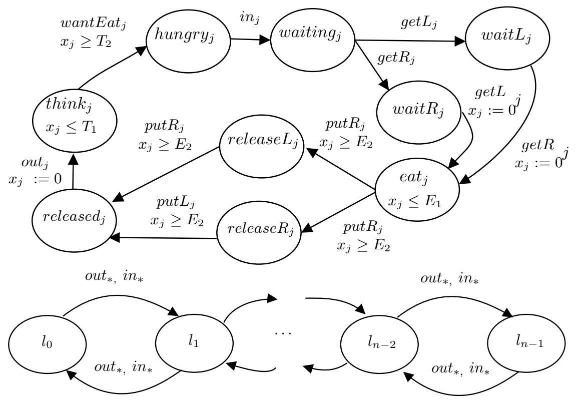

Figure 3: The philosopher (above) and the lackey (below) in the Timed Dining Philosophers benchmark.

computational complexity of the problem). Note that the number of locations of the controller is fixed at 2 in this benchmark: since the property ϕ_2 is already satisfied, there is no need to attempt synthesising a larger module. Moreover, incrementing the bound from an initial minimal value guarantees that an optimal (smallest possible) controller will be obtained, which of course represents a major advantage from a practical standpoint.

Timed Dining Philosophers. Finally, we also include a significantly more involved variant of Dining Philosophers, previously featured in (Kacprzak et al., 2023) and (Arias et al., 2024).

This version, presented in Fig. 3, includes a separate "lackey" agent (to be synthesised), who controls the philosophers' access to the dining room. Moreover, the philosophers' components (given in the partial models specification) are much more complex and contain timing constraints in the form of invariants on several locations, as well as guards on transitions. Some of the latter also reset the clock. As such, this benchmark allows us to better evaluate how the efficiency of our approach is impacted by the continuous time representation in the input partial CAMAS specification. To that end, we consider two additional variants of each test instance: a *reduced* model, in which we make an additional assumption that there are no clocks and no invariants in the lackey automaton, and an *untimed* one, where we also assume no clocks. The results are summarised in Table 3.

Clearly, the results indicate that even partial synthesis remains a major challenge when timing constraints are involved. That is to be expected, as the continuous time representation is the main source of the complexity in the verification and satisfiability checking of timed strategic and temporal logics. Note that while the benchmark was previously used in (Arias et al., 2024), only the untimed case was con-

Table 3: Results for Timed Dining Philosophers.

# philos.	result	model	running time (s)
2	sat	full	50
		reduced	0.25
		untimed	0.06
3	sat	full	memout
		reduced	447
		untimed	16

sidered there in the context of (partial) synthesis for **STCTL**. Thus our method, albeit slower, can be applied to a much broader class of models. Moreover, the restrictions on components to be obtained can be relaxed or made stricter as needed, including for specific locations or transitions that might be known in advance based on the partial specification, allowing to precisely tune it to the needs particular application, and in doing so achieve better efficiency.

We briefly touch upon some aspects of model synthesis for strategic properties, in particular those related to synchronisation between transitions, that remain an interesting subject for further work and could potentially lead to some optimisations in our approach. For instance, depending on the scenario, it may be the case that having no synchronised transitions between certain locations is not a viable option for the resulting system, and as such the IMITATOR model can be simplified accordingly. Moreover, when synthesising a strategy for agents in some coalition A, one could restrict the set of transitions considered for synchronisation only to those "owned" by agents in A, i.e., transitions that are in their local protocols.

5 CONCLUSIONS

We proposed an initial approach to (bounded) satisfiability checking for **TCTL** and its strategic extension **STCTL** under the imperfect information semantics. Using the IMITATOR verifier, we demonstrated the practical feasibility of our method by means of model checking the input formula over a parametric timed automaton (PTA) encoding all possible PTA up to a bounded number of locations.

In the future, we intend to investigate whether this encoding can be improved, which would potentially lead to better scalability and allow for lifting some of the current restrictions on guards and shared transitions. Other ways of tackling **TCTL** and **STCTL** satisfiability also represent potential avenues of further research. In particular, one may consider a full-fledged encoding of the problem to SMT (Klenze et al., 2016; Kacprzak et al., 2020; Niewiadomski et al., 2020; Kacprzak et al., 2021; Kacprzak et al., 2023), or extending the declarative approach of (Arias et al., 2024) to support synthesis of timed components and possibly also full models.

ACKNOWLEDGEMENTS

This work was supported by the PHC Polonium project MoCcA (BPN/BFR/2023/1/00045), CNRS IRP "Le Trójkąt", the ANR-22-CE48-0012 project BISOUS, and by NCBR Poland & FNR Luxembourg under the PolLux/FNR-CORE project SpaceVote (POLLUX-XI/14/SpaceVote/2023).

REFERENCES

Alur, R., Courcoubetis, C., and Dill, D. L. (1993a). Model-Checking in Dense Real-Time. *Inf. Comput.*, 104(1):2–34.

Alur, R. and Dill, D. L. (1990). Automata for Modeling Real-Time Systems. In *Proceedings of ICALP'90*, pages 322–335. Springer.

Alur, R., Henzinger, T., Mang, F., Qadeer, S., Rajamani, S., and Tasiran, S. (1998). MOCHA: Modularity in Model Checking. In *Proceedings of CAV'98*, pages 521–525. Springer.

Alur, R. and Henzinger, T. A. (1993). Real-Time Logics: Complexity and Expressiveness. *Inf. Comput.*, 104(1):35–77.

Alur, R., Henzinger, T. A., and Kupferman, O. (2002). Alternating-time Temporal Logic. *J. ACM*, 49:672–713.

Alur, R., Henzinger, T. A., and Vardi, M. Y. (1993b). Parametric Real-time Reasoning. In *Proceedings of STOC '93*, pages 592–601. ACM.

André, É. (2019). What's Decidable about Parametric Timed Automata? *Int. J. Softw. Tools Technol. Transf.*, 21(2):203–219.

André, É. (2021). IMITATOR 3: Synthesis of timing parameters beyond decidability. In *Proceedings of CAV 2021*, pages 552–565. Springer.

André, É., Bloemen, V., Petrucci, L., and van de Pol, J. (2019). Minimal-Time Synthesis for Parametric Timed Automata. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 211–228. Springer.

André, É., Knapik, M., Penczek, W., and Petrucci, L. (2016). Controlling Actions and Time in Parametric Timed Automata. In *Proceedings of ACSD'16*, pages 45–54. IEEE.

Arias, J., Budde, C. E., Penczek, W., Petrucci, L., Sidoruk, T., and Stoelinga, M. (2020). Hackers vs. Security: ADTrees as Asynchronous Multi-agent Systems. In *Proceedings of ICFEM 2020*, pages 3–19. Springer.

Arias, J., Jamroga, W., Penczek, W., Petrucci, L., and Sidoruk, T. (2023). Strategic (Timed) Computation

Tree Logic. In *Proceedings of AAMAS'23*, pages 382–390. ACM.

Arias, J., Olarte, C., Penczek, W., Petrucci, L., and Sidoruk, T. (2024). Model checking and synthesis for strategic timed CTL using strategies in rewriting logic. In *Proceedings of PPDP 2024*, pages 10:1–10:14. ACM.

Belardinelli, F., Jamroga, W., Kurpiewski, D., Malvone, V., and Murano, A. (2019). Strategy Logic with Simple Goals: Tractable Reasoning about Strategies. In *Proceedings of IJCAI'19*, pages 88–94. ijcai.org.

Bouyer, P. (2009). Model-Checking Timed Temporal Logics. *Electron. Notes Theor. Comput. Sci.*, 231:323–341.

Chen, T. and Lu, J. (2008). Towards the Complexity of Controls for Timed Automata with a Small Number of Clocks. In *Proceedings of FSKD'08*, pages 134–138.

David, A. (2015). Deciding ATL* Satisfiability by Tableaux. In *Proceedings of CADE'15*, pages 214–228. Springer.

Daws, C. and Yovine, S. (1996). Reducing the Number of Clock Variables of Timed Automata. In *Proceedings of RTSS'96*, pages 73–81.

Dima, C. and Tiplea, F. (2011). Model-Checking ATL under Imperfect Information and Perfect Recall Semantics is Undecidable. *CoRR*, abs/1102.4225.

Dubinin, V., Vyatkin, V., and Hanisch, H.-M. (2015). Synthesis of Safety Controllers for Distributed Automation Systems on the Basis of Reverse Safe Net Condition/Event Systems. In *Proceedings of TrustCom'15*, pages 287–292.

Fagin, R., Halpern, J. Y., Moses, Y., and Vardi, M. Y. (1995). *Reasoning about Knowledge*. MIT Press.

Goranko, V. and Shkatov, D. (2009). Tableau-based Decision Procedures for Logics of Strategic Ability in Multiagent Systems. *ACM Trans. Comput. Log.*, 11(1).

Guha, S., Narayan, C., and Arun-Kumar, S. (2014). Reducing clocks in timed automata while preserving bisimulation. In *Proceedings of CONCUR 2014*, pages 527–543. Springer.

Hoek, W. and Wooldridge, M. (2002). Tractable Multiagent Planning for Epistemic Goals. In *Proceedings of AAMAS'02*, pages 1167–1174. ACM Press.

Jamroga, W. (2015). *Logical Methods for Specification and Verification of Multi-Agent Systems*. ICS PAS Publishing House.

Jamroga, W., Masko, L., Mikulski, L., Pazderski, W., Penczek, W., Sidoruk, T., and Kurpiewski, D. (2022). Verification of Multi-Agent Properties in Electronic Voting: A Case Study. In *Proceedings of AiML 2022*, pages 531–556. College Publications.

Jamroga, W., Penczek, W., Sidoruk, T., Dembinski, P., and Mazurkiewicz, A. W. (2020). Towards Partial Order Reductions for Strategic Ability. *JAIR*, 68:817–850.

Kacprzak, M., Niewiadomski, A., and Penczek, W. (2020). SAT-Based ATL Satisfiability Checking. In *Proceedings of KR'20*, pages 539–549.

Kacprzak, M., Niewiadomski, A., and Penczek, W. (2021). Satisfiability Checking of Strategy Logic with Simple Goals. In *Proceedings of KR'21*, pages 400–410.

Kacprzak, M., Niewiadomski, A., Penczek, W., and Zbrzezny, A. (2023). SMT-Based Satisfiability Checking of Strategic Metric Temporal Logic. In *Proceedings of ECAI'23*, pages 1180–1189. IOS Press.

Klenze, T., Bayless, S., and Hu, A. J. (2016). Fast, Flexible, and Minimal CTL Synthesis via SMT. In *Proceedings of CAV'16*, pages 136–156. Springer.

Knapik, M., André, É., Petrucci, L., Jamroga, W., and Penczek, W. (2019). Timed ATL: Forget Memory, Just Count. *J. Artif. Intell. Res.*, 66:197–223.

La Torre, S. and Napoli, M. (2000). A decidable dense branching-time temporal logic. In *Proceedings of FST TCS 2000*, pages 139–150. Springer.

Laroussinie, F., Larsen, K. G., and Weise, C. (1995). From timed automata to logic - and back. In *Proceedings of MFCS'95*, pages 529–539. Springer.

Laroussinie, F., Markey, N., and Oreiby, G. (2006). Model-Checking Timed ATL for Durational Concurrent Game Structures. In *Proceedings of FORMATS'06*, pages 245–259. Springer.

Laroussinie, F., Markey, N., and Schnoebelen, P. (2004). Model Checking Timed Automata with One or Two Clocks. In *Proceedings of CONCUR'04*, pages 387–401. Springer.

Lomuscio, A., Penczek, W., and Qu, H. (2010). Partial Order Reductions for Model Checking Temporal-Epistemic Logics over Interleaved Multi-Agent Systems. *Fundam. Informaticae*, 101(1-2):71–90.

Lomuscio, A. and Raimondi, F. (2006). Model Checking Knowledge, Strategies, and Games in Multi-Agent Systems. In *Proceedings of AAMAS'06*, pages 161–168. ACM.

Meseguer, J. (1992). Conditional Rewriting Logic as a Unified Model of Concurrency. *Theoretical Computer Science*, 96(1):73–155.

Meseguer, J. (2012). Twenty Years of Rewriting Logic. *Journal of Logic and Algebraic Programming*, 81(7-8):721–781.

Niewiadomski, A., Kacprzak, M., Kurpiewski, D., Knapik, M., Penczek, W., and Jamroga, W. (2020). MsATL: A Tool for SAT-Based ATL Satisfiability Checking. In *Proceedings of AAMAS'20*, pages 2111–2113. IFAAMAS.

Penczek, W. and Pólrola, A. (2006). *Advances in Verification of Time Petri Nets and Timed Automata: A Temporal Logic Approach*. Springer.

Petrucci, L., Knapik, M., Penczek, W., and Sidoruk, T. (2019). Squeezing State Spaces of ADTrees. In *Proceedings of ICECCS 2019*, pages 71–80. IEEE.

Schobbens, P. Y. (2004). Alternating-Time Logic with Imperfect Recall. In *Proceedings of LCMAS'03*, pages 1–12.