strategFTO: Untimed Control for Timed Opacity

Étienne André
Université Sorbonne Paris Nord, LIPN, CNRS UMR 7030
Villetaneuse, France

Shapagat Bolat
Engel Lefaucheux
Dylan Marinho
Université de Lorraine, CNRS, Inria, LORIA
Nancy, France

Abstract
We introduce a prototype tool strategFTO addressing the verification of a security property in critical software. We consider a recent definition of timed opacity where an attacker aims to deduce some secret while having access only to the total execution time. The system, here modelled by timed automata, is deemed opaque if for any execution time, there are either no corresponding runs, or both public and private corresponding runs. We focus on the untimed control problem: exhibiting a controller, i.e., a set of allowed actions, such that the system restricted to those actions is fully timed-opaque. We first show that this problem is not more complex than the full timed opacity problem, and then we propose an algorithm, implemented and evaluated in practice.

CCS Concepts: • Security and privacy → Logic and verification; • Theory of computation → Quantitative automata; Verification by model checking.

Keywords: opacity, timing leak, timed automata, security, control, IMITATOR.

ACM Reference Format:
Étienne André, Shapagat Bolat, Engel Lefaucheux, and Dylan Marinho. 2022. strategFTO: Untimed Control for Timed Opacity. In Proceedings of the 8th ACM SIGPLAN International Workshop on Formal Techniques for Safety-Critical Systems (FTSCS ’22), December 07, 2022, Auckland, New Zealand. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3563822.3568013

1 Introduction
We address here the control of timed systems to avoid timing leaks, i.e., the leakage of private information that can be deduced from time. We use as underlying model timed automata (TAs) [1], an extension of finite-state automata with real-valued clocks. Opacity is a key security property requiring that an external user should not be able to deduce whether the execution of a system contains a secret behaviour through its observation. This property was first formalized for labeled transition systems [15], by specifying a subset of secret paths and requiring that, for any secret path, there is a non-secret one with the same observation. Opacity raises challenging research issues such as 1) specifying formally opacity in various frameworks [15, 22], 2) verifying opacity properties [15, 26], and 3) developing mechanisms to design a system satisfying opacity while preserving functionality and performance [10, 16].

Franck Cassez proposed in [17] a first definition of timed opacity asking whether an attacker can deduce a secret by observing a set of observable actions together with their timestamp. He proved that opacity is undecidable for TAs, mainly from the undecidability of the language inclusion problem for TAs [1]. Based on this definition of opacity, some decidable subclasses were proposed, for real-time automata [29, 30] (a severely restricted subclass of TAs with a single clock), or over bounded-time [3].

In [7], we proposed a definition of opacity where the attacker only has access (in addition to the model knowledge) to the system execution time, i.e., the time from the initial location to a given location. The timed opacity problem therefore asks “for which execution times is the attacker unable to deduce whether a private location was visited?” The full timed opacity problem asks whether the system is timed-opaque for all execution times, i.e., the attacker is never able to deduce whether the private location was visited by an execution. We proved in [7] that this latter problem is decidable (in 3EXPTIME), and we proposed a practical algorithm using a parametric version of TAs [2], implemented in IMITATOR [4].

Contribution. If a system is not fully timed-opaque, there may be ways to tune it to enforce opacity. For instance, one could change internal delays, or add some sleep() or Wait() statements in the program (see e.g., [7]). In this paper, we consider a static (untimed) form of control of the system. This indicates whether there is a way of restricting the behavior of users to ensure full timed opacity. With that mindset, we assume the set of actions of the TA is partitioned into a set of controllable actions (that can be disabled) and a set of uncontrollable actions (that cannot be disabled). We address the following goal: exhibit a controller (i.e., a subset of the
We assume a set of controllable actions to be kept in addition to the uncontrollable actions, while other controllable actions are disabled) guaranteeing the system to be fully timed-opaque. We propose an algorithm exhibiting a set of controllers ensuring opacity, implemented into a tool strategFTO, calling IMITATOR [4] for computing suitable opaque execution times, and PoIsOp [9] for additional polyhedra operations.

**Related Works.** It is well known that observing the time taken by a system to finish some operation is a potential way to get information out of it (see e.g., [25]). As such, identifying which information is released by the timing of a system has been studied both from a security and a safety perspective.

From the security point of view, beyond the works related to timed opacity and TAs [3, 7, 17, 29, 30], the notion of non-interference has been widely studied. A first definition of timed non-interference was proposed for TAs in [11, 12]. This notion is extended to PTAs in [6], with a semi-algorithm implemented using IMITATOR [4]. In [20], another notion of timed interference called timed strong non-deterministic non-interference (SNNI) which was based on timed language equivalence between the automaton with hidden low-level actions and the automaton with removed low-level actions was developed. This notion is in some aspects stronger than the opacity notion we consider, and is undecidable. SNNI was adapted in [28] to allow some intentional information leakage and a form of control aimed at ensuring it was presented in [13]. Their framework gives to the attacker more information than the total execution time, and their control differs from ours to include that knowledge.

The **diagnosis** of TAs is one of the dominant research directions aimed at analysing information leakage from a safety perspective. Its goal is to detect, by observing the system, whether some faulty behaviour occurred. As such, it is some form of dual to opacity. Diagnosis was first introduced for TAs in [27]. Diagnosability of a system is shown there to be decidable, though the actual diagnoser may be quite complex (see [14] for subclasses of TAs allowing simpler diagnoser, see also [19] for a summary of the main results on the diagnosis of TAs and [18] for a diagnosability focused control of TAs).

## 2 Preliminaries

We assume a set \( \mathbb{X} = \{x_1, \ldots, x_T\} \) of clocks, i.e., real-valued variables that all evolve over time at the same rate. A clock valuation is a function \( \mu : \mathbb{X} \rightarrow \mathbb{R}_{\geq 0} \). We write \( \overline{0} \) for the clock valuation assigning 0 to all clocks. Given \( d \in \mathbb{R}_{\geq 0}, \mu + d \) denotes the valuation s.t. \((\mu + d)(x) = \mu(x) + d, \) for all \( x \in \mathbb{X} \). Given \( R \subseteq \mathbb{X} \), we define the **reset** of a valuation \( \mu \), denoted by \([\mu]_R\), as follows: \([\mu]_R(x) = 0 \) if \( x \in R \), and \([\mu]_R(x) = \mu(x) \) otherwise.

A clock guard \( g \) is a constraint over \( \mathbb{X} \) defined by a conjunction of inequalities of the form \( x \Rightarrow d, \) with \( d \in \mathbb{Z} \) and \( \Rightarrow \in \{<, \leq, =, \geq, >\} \). Given \( g \), we write \( \mu \models g \) if the expression obtained by replacing each \( x \) with \( \mu(x) \) in \( g \) evaluates to true.

**Definition 2.1 (TA [1]).** A TA \( \mathcal{A} = (\Sigma, L, \ell_0, \ell_{\text{priv}}, \ell_f, \mathbb{X}, I, E) \), where: i) \( \Sigma \) is a finite set of actions, ii) \( L \) is a finite set of locations, iii) \( \ell_0 \in L \) is the initial location, iv) \( \ell_{\text{priv}} \in L \) is the private location, v) \( \ell_f \in L \) is the final location, vi) \( \mathbb{X} \) is a finite set of clocks, vii) \( I \) is the invariant, assigning to every \( \ell \in L \) a clock guard \( I(\ell) \), viii) \( E \) is a finite set of edges \( e = (\ell, g, a, R, \ell') \) where \( \ell, \ell' \in L \) are the source and target locations, \( a \in \Sigma, \forall \ell \subseteq \mathbb{X} \) is a set of clocks to be reset, and \( g \) is a clock guard.

**Example 2.2.** Consider the TA in Fig. 1a, using one clock \( x \). \( \ell_1 \) is the initial location, while we assume that \( \ell_f \) is the final location, i.e., a location in which an attacker can measure the execution time from the initial location. \( \ell_2 \) is the private location, i.e., a secret to be preserved: the attacker should not be able to deduce whether it was visited or not. \( \ell_2 \) has an invariant \( x \leq 3 \) (boxed); other locations invariants are true.

**Definition 2.3 (Semantics of a TA [1]).** Given a TA \( \mathcal{A} = (\Sigma, L, \ell_0, \ell_{\text{priv}}, \ell_f, \mathbb{X}, I, E) \), the semantics of \( \mathcal{A} \) is given by the timed transition system (TTS) \( T(\mathcal{A}) = (S, s_0, \rightarrow) \), with

- \( S = \{(\ell, \mu) \in L \times \mathbb{R}^H_{\geq 0} | [\mu]_I(\ell) = s_0 = (\ell_0, \overline{0})\} \)
- \( \rightarrow \) consists of the discrete and (continuous) delay transition relations: i) discrete transitions: \( (\ell, \mu) \xrightarrow{\epsilon} (\ell', \mu') \), if \((\ell, \mu), (\ell', \mu') \in S \), and there exists \( e = (\ell, g, a, R, \ell') \in E \), such that \( \mu' = [\mu]_g = I(\ell') \), and \( \mu \models g \).
- 2) delay transitions: \( (\ell, \mu) \xrightarrow{d} (\ell, \mu + d) \), with \( d \in \mathbb{R}_{\geq 0} \), if \( \forall d' \in [0, d], (\ell, \mu + d') \in S \).

Moreover we write \( (\ell, \mu) \xrightarrow{(d, e)} (\ell', \mu') \) for a combination of a delay and discrete transition if \( \exists \mu' : (\ell, \mu) \xrightarrow{d} (\ell', \mu') \)

A given a TA \( \mathcal{A} \) with semantics \( (S, s_0, \rightarrow) \), a **run of \( \mathcal{A} \)** is an alternating sequence of states of \( T(\mathcal{A}) \) and pairs of delays and edges starting from the initial state \( s_0 \) of the form \( s_0, (d_0, e_0), s_1, \ldots \) where for all \( i, e_i \in E, d_i \in \mathbb{R}_{\geq 0} \) and \( s_i \xrightarrow{(d_i, e_i)} s_{i+1} \). The **duration of a finite run** \( \rho : s_0, (d_0, e_0), s_1, \ldots, (d_{i-1}, e_{i-1}), (\ell_i, \mu_i) \) is \( d(\rho) = \sum_{0 \leq j \leq i-1} d_j \).

**Timed Opacity Definitions.** We recall here the notion of timed opacity defined in [7].

Given \( \mathcal{A} = (\Sigma, L, \ell_0, \ell_{\text{priv}}, \ell_f, \mathbb{X}, I, E) \) and a run \( \rho \), we say that \( \ell_{\text{priv}} \) is reached on the way to \( \ell_f \) in \( \rho \) if \( p \) is of the form \( (\ell_0, \mu_0), (\ell_d, e_0), (\ell_1, \mu_1), \ldots, (\ell_m, \mu_m), (\ell_d, e_n), \ldots, (\ell_m, \mu_n) \) for some \( m, n \in \mathbb{N} \) such that \( \ell_m = \ell_{\text{priv}}, \ell_n = \ell_f \) and \( \forall 0 \leq i < m - 1, \ell_i \neq \ell_f \). We denote by Reach_{\mathcal{A}}^{\ell_{\text{priv}}} (\ell_f) the set of those runs, and refer to them as private runs. Conversely, we say that \( \ell_{\text{priv}} \) is avoided on the way to \( \ell_f \) in \( \rho \) if \( p \) is of

\[ \text{Reach}_{\mathcal{A}}^{\ell_{\text{priv}}} (\ell_f) \]
the form \((\ell_0, \mu_0), (d_0, e_0), (\ell_1, \mu_1), \cdots, (\ell_n, \mu_n)\) with \(\ell_n = \ell_f\) and \(0 \leq i < n, \ell_i \notin \{\ell_{\text{priv}}, \ell_f\}\). We denote the set of those runs by \(\text{Reach}_{\ell_{\text{priv}}}^A(\ell_f)\), and refer to them as public runs.

While we model the secret behaviour of the system using a private location \(\ell_{\text{priv}}\), note that one could easily adapt these definitions if the secret is, for example, a set of locations, an action (this will be the case in our case study) or the value these definitions if the secret is, for example, a set of locations, an action (this will be the case in our case study) or the value of a variable.

\(\text{DReach}_{\ell_{\text{priv}}}^A\) (resp. \(\text{DReach}_{\ell_{\text{priv}}}^{-\ell_{\text{priv}}}^A\)) is the set of all the durations of the runs for which \(\ell_{\text{priv}}\) is reached (resp. avoided) on the way to \(\ell_f\). Formally: \(\text{DReach}_{\ell_{\text{priv}}}^A = (d \in \mathbb{R}_{\geq 0} \mid \exists \rho \in \text{Reach}_{\ell_{\text{priv}}}^A(\ell_f) \text{ such that } d = \text{dur}(\rho))\) and \(\text{DReach}_{\ell_{\text{priv}}}^{-\ell_{\text{priv}}}^A = \{d \in \mathbb{R}_{\geq 0} \mid \exists \rho \in \text{Reach}_{\ell_{\text{priv}}}^{-\ell_{\text{priv}}}^A(\ell_f) \text{ such that } d = \text{dur}(\rho)\}\).

**Definition 2.4** (full timed opacity). Given a TA \(A\), we say that \(A\) is fully timed-opaque if \(\text{DReach}_{\ell_{\text{priv}}}^A = \text{DReach}_{\ell_{\text{priv}}}^{-\ell_{\text{priv}}}^A\).

That is, a system is fully timed-opaque if, for any execution time \(d\), there exists a run of duration \(d\) that reaches \(\ell_f\) after going through \(\ell_{\text{priv}}\) iff there exists another run of duration \(d\) that reaches \(\ell_f\) without going through \(\ell_{\text{priv}}\). Hence, the attacker cannot deduce from the execution time whether \(\ell_{\text{priv}}\) was visited or not.

**Example 2.5.** Consider again the TA in Fig. 1a. Recall that \(\ell_2\) is the private location. We have \(\text{DReach}_{\ell_{\text{priv}}}^A = [1, 5]\) and \(\text{DReach}_{\ell_{\text{priv}}}^{-\ell_{\text{priv}}}^A = [1, 3] \cup [4, 4] \cup (5, +\infty)\). Since \(\text{DReach}_{\ell_{\text{priv}}}^A \neq \text{DReach}_{\ell_{\text{priv}}}^{-\ell_{\text{priv}}}^A\), the system is not fully timed-opaque.

## 3 Untimed Control for Full Timed Opacity

In this section, we introduce an untimed control for controlling timed opacity. We assume \(\Sigma = \Sigma_c \cup \Sigma_u\) where \(\Sigma_c\) (resp. \(\Sigma_u\)) denote controllable (resp. uncontrollable) actions.

A (static, untimed) strategy of a TA \(A\) is a set of actions \(\sigma \subseteq \Sigma\) that contains at least all uncontrollable actions (i.e., \(\Sigma_u \subseteq \sigma \subseteq \Sigma\)). A strategy induces a restriction of \(A\) where only the edges labeled by actions of \(\sigma\) are allowed:

**Definition 3.1** (Controlled TA). Given \(A = (\Sigma, L, \ell_0, \ell_{\text{priv}}, \ell_f, X, I, E)\) with \(\Sigma = \Sigma_u \cup \Sigma_c\) and a strategy \(\sigma \subseteq \Sigma\), the control of \(A\) using \(\sigma\) is the TA \(A' = \text{Control}(A, \sigma) = (\sigma, L, \ell_0, \ell_{\text{priv}}, \ell_f, X, I, E')\) where \(E' = \{(\ell, g, a, R, \ell') \in E \mid a \in \sigma\}\).

**Example 3.2.** Consider again the TA \(A\) in Fig. 1a. Fix \(\sigma = \{u, a\}\). Then \(\text{Control}(A, \sigma)\) is in Fig. 1b.

Strategies represent some modifications of the system that can be implemented to ensure full timed opacity.

**Definition 3.3** (fully timed-opaque strategy). A strategy \(\sigma\) is fully timed-opaque if \(\text{Control}(A, \sigma)\) is fully timed-opaque.

A strategy (even a maximal one) might achieve full timed opacity by blocking all runs (both private or public) from reaching the target. If reaching the target means completing a task, this might not be something one would desire. We call a strategy allowing to reach the target for at least some durations an effective strategy.

We define two slightly different problems: taking a TA \(A\) as input, the full timed (resp. effective full time) opacity control emptiness problem asks whether the set of fully (resp. effective fully) timed-opaque strategies for \(A\) is empty.

Note that, due to the presence of uncontrollable actions, the first problem (full timed opacity control emptiness) is not trivial. (If uncontrollable actions were not part of our definitions, choosing \(\sigma = \emptyset\) would always yield an acceptable fully timed-opaque strategy.)

We will also refine those problems by considering a notion of maximal (i.e., most permissive) strategy w.r.t. full timed opacity based on the number of actions belonging to the strategy: given \(A\), a fully timed-opaque strategy \(\sigma\) is maximal if \(\forall \sigma'\), if \(\sigma'\) is fully timed-opaque then \(|\sigma'| \leq |\sigma|\).

We define similarly minimal strategies (least permissive, i.e.,
disabling as many actions as possible) as well as maximal (resp. minimal) effective fully timed-opaque strategies, i.e., the set of largest (resp. smallest) effective fully timed-opaque strategies.

**Example 3.4.** Consider again the TA $A$ in Fig. 1a. Assume $\Sigma_u = \{u\}$ and $\Sigma_c = \{a, b, c, d, e, f\}$. Fix $\sigma_1 = \{u, b, c\}$. We have $DReach^{\text{priv}}(\text{Control}(A, \sigma_1)) = \{2, 5\}$ while $DReach^{\text{priv}}(\text{Control}(A, \sigma_2)) = \{4, 4\}$; therefore, $\sigma_1$ is not fully timed-opaque. Now fix $\sigma_2 = \{u, a, f\}$. We have $DReach^{\text{priv}}(\text{Control}(A, \sigma_2)) = DReach^{\text{priv}}(\text{Control}(A, \sigma_2)) = \{1, 3\}$; therefore, $\sigma_2$ is fully timed-opaque.

In fact, it can be shown that the set of effective fully timed-opaque strategies for $A$ is $\{\{u, a\}, \{u, a, e\}, \{u, a, f\}\}$; therefore, $\{u, a\}$ is the only minimal strategy, while $\{u, a, e\}, \{u, a, f\}$ are the two maximal strategies. In addition, $\{u, f\}$ is an example of a strategy that is not effective, as $T_f$ is always unreachable, whether $t_{\text{priv}}$ is visited or not.

**Proposition 3.5** (complexity). One can compute the set of fully timed-opaque strategies over a TA $A$ in $3\text{EXPTIME}$. 

**Proof.** The full timed opacity decision problem (i.e., checking if a given TA is fully timed-opaque) is decidable for TAs in (at most) $3\text{EXPTIME}$ [7]. Moreover, reachability of the final state can be decided in $\text{PSPACE}$ [1]. Thus, for any given strategy, one can check in triple exponential time whether it is (effective) fully timed-opaque.

Computing the list of (effective) fully timed-opaque strategies can be done naively by testing each possible strategy one by one and keeping the ones that satisfy the property. As there is an exponential number of possible strategies and repeating exponentially many times a $3\text{EXPTIME}$ algorithm remains in $3\text{EXPTIME}$, this algorithm is in $3\text{EXPTIME}$.

As a corollary of the above, the (effective) full timed opacity control emptiness problem is in $3\text{EXPTIME}$ as well. More precisely, the above proof establishes that the complexity class of the (effective) full timed opacity control emptiness problem is the maximum between $\text{PSPACE}$ and the complexity of the full timed opacity problem. As the latter is $\text{PSPACE}$-hard (being trivially harder than reachability), the two problems lie in the same complexity class. From a theoretical point of view, one thus cannot do better than the naive enumeration approach described here to solve the control problem.

Finding the maximal (resp. minimal) strategies can be done slightly more efficiently by starting from the set with every (resp. no) controllable action and enumerating the potential strategies by decreasing (resp. increasing) order as one could then potentially stop before full enumeration. In the worst case, this will however have the same complexity as the full enumeration.

### 4 Implementation and Experiments

We implemented our strategy generation in strategFTO, an entirely automated open-source tool written in Java. Our tool iteratively constructs strategies, then checks full timed opacity by computing the private and public execution times and by checking their equality.

The exhibition of these execution times ($DReach^{\text{priv}}(A)$ and $DReach^{\text{priv}}(A)$) is done in our implementation by an automated model modification (following the procedure described in [7]), but which was not entirely automated in [7] followed by a synthesis problem using a parametric extension of TAs [2]. The synthesis of the execution times itself is done by a call to an external tool—IMITATOR 3.3 “Cheese Caramel au beurre sale” [4]. strategFTO then checks whether both sets of execution times are equal; this is done by a call to another external tool—POLYOr 1.2, that performs polyhedral operations using PPL [9].

**Algorithms.** We implement not only the exhibition of all timed-opaque strategies (denoted by synthCtrl($A$)), but also the following variants: i) synthMaxCtrl($A$): synthesize all maximal strategies for $A$; ii) synthMinCtrl($A$): synthesize all minimal strategies; iii) witnessMaxCtrl($A$): witness one maximal strategy; iv) witnessMinCtrl($A$): witness one minimal strategy. We implemented these other algorithms by changing the exploration order of the strategies, and/or by triggering immediate termination upon the first exhibition of a strategy.

**Input Model.** The input TA model is given in the IMITATOR input syntax; while we presented a restricted setting in this paper for sake of clarity, our implementation in strategFTO is much more permissive, by allowing significant extensions of TAs with global (integer or Boolean) variables, multiple automata with synchronization, multi-rate clocks (including stopwatches), etc.

#### 4.1 Proof of Concept Benchmark

As a proof of concept, we consider the TA model of an ATM (given in Fig. 2). The idea is that (as per our definition of timed opacity) the attacker only has access to the execution time, i.e., the time from the beginning of the program to reaching the end state. The secret is whether the ATM user has actually obtained cash (action takeCash). The TA uses two clocks: $x$ for “local” actions, and $y$ for a global time measurement. First, the user starts the process (action start), then the ATM displays a welcome screen for 3 time units, followed by another screen requesting the password (action askPwd). Then, the user can submit a correct (action correctPwd) or incorrect (incorrectPwd) password; if no password is input

---

2Source code is at https://github.com/DylanMarinho/Controlling-TA. Models and experiment results are available at 10.5281/zenodo.7181848.

3https://github.com/etienneandre/PolyOp

4strategFTO allows not only private locations, but also actions.
within 10 time units, the system moves to a cancelling phase. The same happens if 3 incorrect passwords have been input. After inputting the correct password, the user has the choice between a fixed-amount quick withdrawal (quickWdraw), a normal withdrawal (normalWithdraw) or a balance request (reqBalance).

The quick withdrawal triggers a 15-time unit preparation followed by the availability of the money, which the user can take immediately (action takeCash), thus terminating the procedure. If the user does not take the money, the system moves to the cancelling phase.

The normal withdrawal asks the user to input the desired amount; similar to the password, after 3 wrong amounts (action incorrectAmount), or upon timeout, the system moves to cancelling phase. After the user retrieves cash (action takeCash), they are asked whether they would like to perform another operation; if so (action restart), the system goes back to the choice location. Otherwise (action pressFinish), or unless a 10-time unit timeout is reached, the system moves to the terminating location. The balance request triggers the balance display, from which the user can immediately terminate the process (action pressOK), or go back to the choice menu.

The rationale is that, in the regular terminating and cancelling phases, the ATM terminates after constant time (invariant $y \leq 100$), avoiding leaking information. However, some actions may lead to quicker termination (quick withdrawal) or slower termination (multiple choices).

The uncontrollable actions are most of the user actions: correctAmount, incorrectAmount, correctPwd, incorrectPwd, pressFinish, takeCash. The controllable actions are the system actions (askPwd, start, finish) and some of the users actions that can be controlled by disabling the associated choice (reqBalance, pressOK, quickWdraw, restart).

### 4.2 Experiments

We exhibit in Table 1 controllers for our Fig. 2 benchmark computed by strategFTO, for all algorithms. For space concern, we tabulate the actions to disable; the strategy is therefore $\Sigma$ minus these actions. Also note that, for witnessMaxCtrl and witnessMinCtrl, the order in which we compute the subsets of $\Sigma$ has an impact on the result, as the algorithm stops as soon as one strategy is found. According to Table 1, the maximal strategies (i.e., the most permissive, disabling the least number of actions) are to disable either restart and pressOK, or restart and reqBalance. This is natural, as restart allows the user to restart a second operation, thus violating the constant-time nature of Fig. 2, while pressOK and reqBalance, if enabled together, allow a quick exit, shorter than a cash withdrawal operation—thus giving hint to the attacker that the takeCash secret did not occur.

**Scalability.** Then, we test the scalability of strategFTO w.r.t. the number of actions. We modify Fig. 2 by adding an increasingly large numbers of controllable actions; these actions do not play a role in the control (we basically add unguarded self-loops) but they will impact the computation.

![Figure 2. ATM benchmark](image-url)
Table 1. Strategy synthesis for Fig. 2

| Actions to disable | synthMinCtrl | witnessMinCtrl | synthMaxCtrl | witnessMaxCtrl | synthCtrl |
|--------------------|--------------|----------------|--------------|----------------|-----------|
| restart, pressOK   | 🟢            | 🟢              | 🟢            | 🟢              | 🟢        |
| restart, pressOK, quickWdraw | 🟢            | 🟢              | 🟢            | 🟢              | 🟢        |
| restart, pressOK, reqBalance | 🟢            | 🟢              | 🟢            | 🟢              | 🟢        |

Figure 3. Execution times for scalability (in seconds; TO set at 1,800 s)

5 Conclusion

We introduced a prototype tool strategFTO implementing an algorithm to exhibit strategies to guarantee the full timed opacity of a system modeled by a timed automaton where the attacker only has access to the computation time. Even though relying on a simple enumeration of the subsets, our tool strategFTO shows good performance for synthesizing maximal or minimal strategies, with very reasonable times, even for several dozens of controllable actions.

Future Works. We plan to further optimize our implementation by maintaining a set of non-effective strategies, i.e., for which \( t_f \) is unreachable: any strategy strictly included into a known non-effective strategy will necessarily be non-effective too, and therefore no full timed-opacity analysis is needed for this strategy. An option to efficiently represent this strategies set could be to store it using BDDs.

We also plan to strengthen strategies so that their choice may depend on how long has passed since the start of the execution. As these strategies still need a finite representation to be handled, this requires establishing exactly what strategies need to remember to chose optimally.

Our ultimate goal will be to extend timed automata to parametric timed automata, and use automated parameter synthesis techniques (e.g., [5, 8, 23]), with a parametric timed controller [21, 24].

Acknowledgments

This work is partially supported by the ANR-NRF French-Singaporean research program ProMiS (ANR-19-CE25-0015 / 2019 ANR NRF 0092) and the ANR research program BisoUS.

Experiments presented in this paper were carried out using the Grid’5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several universities as well as other organizations (see https://www.grid5000.fr).
