Complexity of Verification and Synthesis of Threshold Automata *

A. R. Balasubramanian, Javier Esparza, and Marijana Lazić

Technical University of Munich, Germany

Abstract. Threshold automata are a formalism for modeling and analyzing fault-tolerant distributed algorithms, recently introduced by Konnov, Veith, and Widder, describing protocols executed by a fixed but arbitrary number of processes. We conduct the first systematic study of the complexity of verification and synthesis problems for threshold automata. We prove that the coverability, reachability, safety, and liveness problems are NP-complete, and that the bounded synthesis problem is \( \Sigma^p_2 \)-complete. A key to our results is a novel characterization of the reachability relation of a threshold automaton as an existential Presburger formula. The characterization also leads to novel verification and synthesis algorithms. We report on an implementation, and provide experimental results.

Keywords: Threshold automata, distributed algorithms, parameterized verification

1 Introduction

Many concurrent and distributed systems consist of an arbitrary number of communicating processes. Parameterized verification investigates how to prove them correct for any number of processes [1].

Parameterized systems whose processes are indistinguishable and finite state are often called replicated systems. A global state of a replicated system is completely determined by the number of processes in each state. Models of replicated systems differ in the communication mechanism between processes. Vector Addition Systems (VAS) and their extensions [12,8,10,2] can model rendez-vous, multiway synchronization, global resets and broadcasts, and other mechanisms. The decidability and complexity of their verification problems is well understood [1,11,25,12].

Transition guards of VAS-based replicated systems are local: Whether a transition is enabled or not depends only on the current states of a fixed number of processes, independent of the total number of processes. Konnov et al. observed in [16] that local guards cannot model fault-tolerant distributed algorithms. Indeed, in such algorithms often a process can only make a step if it has received

* This project has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme under grant agreement No 787367 (PaVeS).
a message from a *majority* or some fraction of the processes. To remedy this, they introduced *threshold automata*, a model of replicated systems with shared-variable communication and *threshold guards*, in which the value of a global variable is compared to an affine combination of the total numbers of processes of different types. In a number of papers, Konnov et al. have developed and implemented verification algorithms for safety and liveness of threshold automata \[10,17,19,15,18\]. Further, Kukovec et al. have obtained decidability and undecidability results \[20\] for different variants of the model. However, contrary to the VAS case, the computational complexity of the main verification problems has not yet been studied.

We conduct the first systematic complexity analysis of threshold automata. In the first part of the paper we show that the parameterized coverability and reachability problems are NP-complete. Parameterized coverability asks if some configuration reachable from some initial configuration puts at least one process in a given state, and parameterized reachability asks if it puts processes in *exactly* a given set of states, leaving all other states unpopulated. The NP upper bound is a consequence of our main result, showing that the reachability relation of threshold automata is expressible in existential Presburger arithmetic. In the second part of the paper we apply this expressibility result to prove that the model checking problem of Fault-Tolerant Temporal Logic (ELTL\textsubscript{FT}) \[19\] is NP-complete, and that the problem of synthesizing the guards of a given automaton, studied in \[22\], is \(\Sigma_2^p\) complete. The last part of the paper reports on an implementation of our novel approach to the parameterized (safety and liveness) verification problems. We show that it compares favorably to ByMC, the tool developed in \[18\].

Due to lack of space most proofs have been moved to the Appendix.

## 2 Threshold Automata

We introduce threshold automata, illustrating the definitions on the example of Figure 2, a model of the Byzantine agreement protocol of Figure 1.

**Environments.** Threshold automata are defined relative to an *environment* \(\text{Env} = (\Pi, RC, N)\), where \(\Pi\) is a set of *parameters* ranging over \(\mathbb{N}_0\), \(RC \subseteq \mathbb{N}_0^{\Pi}\) is a *resilience condition* expressible as an integer linear formula, and \(N : RC \rightarrow \mathbb{N}_0\) is a linear function. Intuitively, a valuation of \(\Pi\) determines the number of processes of different kinds (e.g., faulty) executing the protocol, and \(RC\) describes the admissible combinations of parameter values. Finally, \(N\) associates to each admissible combination, the number of copies of the automaton that are going to run in parallel, or, equivalently, the number of processes explicitly modeled. In a Byzantine setting, faulty processes behave arbitrarily, and so we do not model them explicitly; in this case, the system consists of one copy of the automaton for every correct process. In the crash fault model, processes behave correctly until they crash, they must be modeled explicitly, and the system has a copy of the automaton for each process, faulty or not.
In the threshold automaton of Figure 2, the parameters are \( n, f \), and \( t \), describing the number of processes, the number of faulty processes, and the maximum possible number of faulty processes, respectively. The resilience condition is the set of triples \((i_n,i_f,i_i)\) such that \(i_n/3 > i_f > i_i\); abusing language, we identify it with the constraint \(n/3 > t \geq f\). The function \(N\) is given by \(N(n,t,f) = n - f\), which is the number of correct processes.

**Example 1.** In the threshold automaton of Figure 2, the parameters are \( n, f, \) and \( t \), describing the number of processes, the number of faulty processes, and the maximum possible number of faulty processes, respectively. The resilience condition is the set of triples \((i_n,i_f,i_i)\) such that \(i_n/3 > i_f > i_i\); abusing language, we identify it with the constraint \(n/3 > t \geq f\). The function \(N\) is given by \(N(n,t,f) = n - f\), which is the number of correct processes.

**Threshold automata.** A threshold automaton over an environment \(Env\) is a tuple \(TA = (\mathcal{L}, \mathcal{I}, \Gamma, \mathcal{R})\), where \(\mathcal{L}\) is a nonempty, finite set of local states (or locations), \(\mathcal{I} \subseteq \mathcal{L}\) is a nonempty subset of initial locations, \(\Gamma\) is a set of global variables ranging over \(\mathbb{N}_0\), and \(\mathcal{R}\) is a set of transition rules (or just rules), formally described below.

A transition rule (or just a rule) is a tuple \(r = (\text{from}, \text{to}, \varphi, u)\), where \(\text{from}\) and \(\text{to}\) are the source and target locations, \(\varphi: \Pi \cup \Gamma \rightarrow \{\text{true}, \text{false}\}\) is a conjunction of threshold guards, and \(u: \Gamma \rightarrow \{0,1\}\) is an update. We often let \(r, \text{from}, \text{to}, \varphi, r, u\) denote the components of \(r\). Intuitively, \(r\) states that a process can move from \(\text{from}\) to \(\text{to}\) if the current values of \(\Pi\) and \(\Gamma\) satisfy \(\varphi\), and when it moves it updates the current valuation \(g\) of \(\Gamma\) by performing the up-

```plaintext
1 var myval, ∈ {0, 1}
2 var accept, ∈ {false, true} ← false
3 while true do (in one atomic step)
4   if myval, = 1
5     and not sent ECHO before
6     then send ECHO to all
7   if received ECHO from at least 
8     \(t + 1\) distinct processes
9     and not sent ECHO before
10    then send ECHO to all
11   if received ECHO from at least 
12     \(n - t\) distinct processes
13    then accept, ← true
14 od
```

**Fig. 2.** Threshold automaton modeling the body of the loop in the protocol from Fig. 1. Symbols \(\gamma_1, \gamma_2\) stand for the threshold guards \(x \geq (t+1) - f\) and \(x \geq (n-t) - f\), where \(n\) and \(t\) are as in Fig. 1 and \(f\) is the actual number of faulty processes. The shared variable \(x\) models the number of ECHO messages sent by correct processes. Processes with \(\text{myval}, = b\) (line 1) start in location \(\ell_b\) (in green). Rules \(r_1\) and \(r_2\) model sending ECHO at lines 7 and 12. The self-loop rules \(sl_1, \ldots, sl_3\) are stuttering steps.
date $g := g + u$. Since all components of $u$ are nonnegative, the values of global variables never decrease. A threshold guard $\varphi$ has one of the following two forms:

- $x \geq a_0 + a_1 \cdot p_1 + \ldots + a_{|\Pi|} \cdot p_{|\Pi|}$, called a rise guard, or
- $x < a_0 + a_1 \cdot p_1 + \ldots + a_{|\Pi|} \cdot p_{|\Pi|}$, called a fall guard,

where $x \in \Gamma$ is a shared variable, $p_1, \ldots, p_{|\Pi|} \in \Pi$ are the parameters, and $a_0, a_1, \ldots, a_{|\Pi|} \in \mathbb{Q}$ are rational coefficients. Since global variables are initialized to 0, and they never decrease, once a rise (fall) guard becomes true (false) it stays true (false). We call this property monotonicity of guards. We let $\Phi^{\text{rise}}$, $\Phi^{\text{fall}}$, and $\Phi$ denote the sets of rise guards, fall guards, and all guards of TA.

**Example 2.** The rule $r_2$ of Figure 2 has $\ell_0$ and $\ell_2$ as source and target locations, $x \geq (t + 1) - f$ as guard, and the number 1 as update (there is only one shared variable, which is increased by one).

**Configurations and transition relation.** A configuration of TA is a triple $\sigma = (\kappa, g, p)$ where $\kappa : L \to \mathbb{N}_0$ describes the number of processes at each location, and $g \in \mathbb{N}_0^{[L]}$ and $p \in RC$ are valuations of the global variables and the parameters. In particular, $\sum_{\ell \in L} \kappa(\ell) = N(p)$ always holds. A configuration is initial if $\kappa(\ell) = 0$ for every $\ell \notin I$, and $g = 0$. We often let $\sigma, \kappa, g, p$ denote the components of $\sigma$.

A configuration $\sigma = (\kappa, g, p)$ enables a rule $r = (\text{from}, \text{to}, \varphi, u)$ if $\kappa(\text{from}) > 0$, and $(g, p)$ satisfies the guard $\varphi$, i.e., substituting $g(x)$ for $x$ and $p(p_i)$ for $p_i$ in $\varphi$ yields a true expression, denoted by $\sigma \models \varphi$. If $\sigma$ enables $r$, then TA can move from $\sigma$ to the configuration $r(\sigma) = (\kappa', g', p')$ defined as follows: (i) $p' = p$, (ii) $g' = g + u$, and (iii) $\kappa = \kappa + v_r$, where $v_r(\text{from}) = -1$, $v_r(\text{to}) = +1$, and $v_r = 0$ otherwise. We let $\sigma \rightarrow r(\sigma)$ denote that TA can move from $\sigma$ to $r(\sigma)$.

**Schedules and paths.** A schedule is a (finite or infinite) sequence of rules. A schedule $\tau = r_1, \ldots, r_m$ is applicable to configuration $\sigma_0$ if there is a sequence of configurations $\sigma_0, \ldots, \sigma_m$ such that $\sigma_i = r_i(\sigma_{i-1})$ for $1 \leq i \leq m$, and we define $\tau(\sigma_0) := \sigma_m$. We let $\sigma \rightarrow^* \sigma'$ denote that $\tau(\sigma) = \sigma'$ for some schedule $\tau$, and say that $\sigma'$ is reachable from $\sigma$. Further we let $\tau \cdot \tau'$ denote the concatenation of two schedules $\tau$ and $\tau'$, and, given $\mu \geq 0$, let $\mu \cdot \tau$ the concatenation of $\tau$ with itself $\mu$ times.

A path or run is a finite or infinite sequence $\sigma_0, r_1, \sigma_1, \ldots, r_k, \sigma_k, \ldots$ of alternating configurations and rules such that $\sigma_i = r_i(\sigma_{i-1})$ for every $r_i$ in the sequence. If $\tau = r_1, \ldots, r_{|\tau|}$ is applicable to $\sigma_0$, then we let $\text{path}(\sigma_0, \tau)$ denote the path $\sigma_0, r_1, \sigma_1, \ldots, r_{|\tau|}, \sigma_{|\tau|}$ with $\sigma_i = r_i(\sigma_{i-1})$, for $1 \leq i \leq |\tau|$. Similarly, if $\tau$ is an infinite schedule. Given a path $\text{path}(\sigma, \tau)$, the set of all configurations in the path is denoted by $\text{Cfgs}(\sigma, \tau)$.

### 3 Coverability and Parameterized Coverability

We say that configuration $\sigma$ covers location $\ell$ if $\sigma.\kappa(\ell) > 0$. We consider the following two coverability questions in threshold automata:
Definition 1 ((Parameterized) coverability). The coverability problem consists of deciding, given a threshold automaton $TA$, a location $\ell$ and an initial configuration $\sigma_0$, if some configuration reachable from $\sigma_0$ covers $\ell$. The parameterized coverability problem consists of deciding, given $TA$ and $\ell$, if there is an initial configuration $\sigma_0$ and a configuration reachable from $\sigma_0$ that covers $\ell$.

Sometimes we also speak of the non-parameterized coverability problem, instead of the coverability problem, to avoid confusion. We show that both problems are NP-hard, even when the underlying threshold automaton is acyclic. In the next section, we show that the reachability and parameterized reachability problems (which subsume the corresponding coverability problems) are both in NP.

Theorem 1. Parameterized coverability in threshold automata is NP-hard, even for acyclic threshold automata with only constant guards (i.e., guards of the form $x < a$ and $x \geq a_0$).

Proof. (Sketch.) We prove NP-hardness of parameterized coverability by a reduction from 3-SAT. The reduction is as follows: (See Figure 3 for an illustrative example). Let $\varphi$ be a 3-CNF formula with variables $x_1, \ldots, x_n$. For every variable $x_i$ we will have two shared variables $y_i$ and $\bar{y}_i$. For every clause $C_j$, we will have a shared variable $c_j$. Intuitively, each process begins at some state $\ell_i$ and then moves to either $T_i$ or $\bot_i$ by firing either $(\ell_i, T_i, \bar{y}_i < 1, y_i++)$ or $(\ell_i, \bot_i, y_i < 1, \bar{y}_i++)$ respectively. Moving to $T_i$ (resp.) means that the process has guessed the value of the variable $x_i$ to be true (false resp). Once it has chosen a truth value, it then increments the variables corresponding to all the clauses which it satisfies and moves to a location $\ell_{mid}$. If it happens that all the guesses were correct, a final rule gets unlocked and processes can move from $\ell_{mid}$ to $\ell_F$. The key property we need to show is that if some process moves to $T_i$ then no other process can move to $\bot_i$ (and vice versa). This is indeed the case because if a process moves to $T_i$ from $\ell_i$, it would have fired the rule $(\ell_i, \top_i, \bar{y}_i < 1, y_i++)$ which increments the shared variable $y_i$, and so falsifies the guard of the corresponding rule $(\ell_i, \bot_i, y_i < 1, \bar{y}_i++)$, and therefore no process can fire it. Similarly, if $(\ell_i, \bot_i, y_i < 1, \bar{y}_i++)$ is fired, no process can fire $(\ell_i, \top_i, \bar{y}_i < 1, y_i++)$.

A modification of the same construction proves

Theorem 2. The coverability problem is NP-hard even for acyclic threshold automata with only constant rise guards (i.e., guards of the form $x \geq a_0$).

Constant rise guards. Theorem 2 puts strong constraints to the class of TAs for which parameterized coverability can be polynomial, assuming $P \neq NP$. We identify an interesting polynomial case.

Definition 2. An environment $Env = (\Pi, RC, N)$ is multiplicative for a TA if for every $\mu \in \mathbb{N}_{\geq 0}$ (i) for every valuation $p \in RC$ we have $\mu \cdot p \in RC$ and $N(\mu \cdot p) = \mu \cdot N(p)$, and (ii) for every guard $\varphi := x \odot a_0 + a_1p_1 + a_2p_2 + \ldots a_kp_k$ in $TA$ (where $\odot \in \{\geq, <\}$), if $(y, q_1, q_2, \ldots, q_k)$ is a (rational) solution to $\varphi$ then $(\mu \cdot y, \mu \cdot q_1, \ldots, \mu \cdot q_k)$ is also a solution to $\varphi$.
Multiplicativity is a very mild condition. To the best of our knowledge, all algorithms discussed in the literature, and all benchmarks of [19], have multiplicative environments. For example, in Figure 2, if the resilience condition \( t < n/3 \) holds for a pair \((\mu, n)\), then it also holds for \((\mu \cdot n, \mu \cdot t)\); similarly, the function \( N(n, t, f) = n - f \) also satisfies \( N(\mu \cdot n, \mu \cdot t, \mu \cdot f) = \mu \cdot n - \mu \cdot f = \mu \cdot N(n, t, f) \). Moreover, if \( x \geq t + 1 - f \) holds in \( \sigma \), then we also have \( \mu \cdot x \geq \mu \cdot t + 1 - \mu \cdot f \) in \( \mu \cdot \sigma \). Similarly for the other guard \( x \geq n - t - f \).

This property allows us to reason about multiplied paths in large systems. Namely, condition (ii) from Definition 2 yields that if a rule is enabled in \( \sigma \), it is also enabled in \( \mu \cdot \sigma \). This plays a crucial role in Section 5 where we need the fact that a counterexample in a small system implies a counterexample in a large system.

**Theorem 3.** Parameterized coverability of threshold automata over multiplicative environments with only constant rise guards is P-complete.

**Proof.** (Sketch.) P-hardness is proved by giving a logspace-reduction from the Circuit Value problem ([21]) which is well known to be P-hard. In the following, we sketch the proof of inclusion in P.

Let \( TA = (L, \mathcal{I}, I, R) \) be a threshold automaton over a multiplicative environment \( Env = (\Pi, RC, N) \) such that the guard of each transition in \( R \) is a constant rise guard. We construct the set \( \hat{L} \) of locations that can be reached by at least one process, and the set of transitions \( \hat{R} \) that can occur, from at least one initial configuration. We initialize two variables \( X_L \) and \( X_R \) by \( X_L := \mathcal{I} \) and \( X_R := \emptyset \), and repeatedly update them until a fixed point is reached, as follows:

- If there exists a rule \( r = (\ell, \ell', \text{true}, u) \in R \setminus X_R \) such that \( \ell \in X_L \), then set \( X_L := X_L \cup \{\ell'\} \) and \( X_R := X_R \cup \{r\} \).
– If there exists a rule \( r = (\ell, \ell', (\wedge_{1 \leq i \leq q} x_i \geq c_i), u) \in \mathcal{R} \setminus \mathcal{X}_R \) such that \( \ell \in \mathcal{X}_L \), and there exists rules \( r_1, r_2, \ldots, r_q \) such that each \( r_i = (\ell_i, \ell'_i, \varphi_i, u_i) \in \mathcal{X}_R \) and \( u_i[x_i] > 0 \), then set \( \mathcal{X}_L := \mathcal{X}_L \cup \{\ell'\} \) and \( \mathcal{X}_R := \mathcal{X}_R \cup \{r\} \).

We prove in the Appendix that after termination \( \mathcal{X}_L = \hat{\mathcal{L}} \) holds. Intuitively, multiplicativity ensures that if a reachable configuration enables a rule, there are reachable configurations from which the rule can occur arbitrarily many times. This shows that any path of rules constructed by the algorithm is executable.

4 Reachability

We now consider reachability problems for threshold automata. Formally, we consider the following two versions of the reachability problem:

**Definition 3 ((Parameterized) reachability).** The reachability problem consists of deciding, given a threshold automaton \( \mathcal{TA} \), two sets \( \mathcal{L}_0, \mathcal{L}_>0 \) of locations, and an initial configuration \( \sigma_0 \), if some configuration \( \sigma \) reachable from \( \sigma_0 \) satisfies \( \sigma.\kappa(\ell) = 0 \) for every \( \ell \in \mathcal{L}_0 \) and \( \sigma.\kappa(\ell) > 0 \) for every \( \ell \in \mathcal{L}_>0 \). The parameterized reachability problem consists of deciding, given \( \mathcal{TA} \) and \( \mathcal{L}_0, \mathcal{L}_>0 \), if there is an initial configuration \( \sigma_0 \) such that some \( \sigma \) reachable from \( \sigma_0 \) satisfies \( \sigma.\kappa(\ell) = 0 \) for every \( \ell \in \mathcal{L}_0 \) and \( \sigma.\kappa(\ell) > 0 \) for every \( \ell \in \mathcal{L}_>0 \).

Notice that the reachability problem clearly subsumes the coverability problem and hence, in the sequel, we will only be concerned with proving that both problems are in NP. This will be a consequence of our main result, showing that the reachability relation of threshold automata can be characterized as an existential formula of Presburger arithmetic. This result has several other consequences. In Section 5 we use it to give a new model checking algorithm for the fault-tolerant logic of [19]. In Section 7 we report on an implementation whose runtime compares favorably with previous tools.

**Reachability relation as an existential Presburger formula.** Fix a threshold automaton \( \mathcal{TA} = (\mathcal{L}, \mathcal{I}, \Gamma, \mathcal{R}) \) over an environment \( \text{Env} \). We construct an existential Presburger arithmetic formula \( \phi_{\text{reach}} \) with \( (2|\mathcal{L}| + 2|\Gamma| + 2|\Pi|) \) free variables such that \( \phi_{\text{reach}}(\sigma, \sigma') \) is true iff \( \sigma' \) is reachable from \( \sigma \).

Let the context of a configuration \( \sigma \), denoted by \( \omega(\sigma) \), be the set of all rise guards that evaluate to true and all fall guards that evaluate to false in \( \sigma \). Given a schedule \( \tau \), we say that the path \( \text{path}(\sigma, \tau) \) is steady if all the configurations it visits have the same context. By the monotonicity of the guards of threshold automata, \( \text{path}(\sigma, \tau) \) is steady iff its endpoints have the same context, i.e., iff \( \omega(\sigma) = \omega(\tau(\sigma)) \). We have the following proposition:

**Proposition 1.** Every path of a threshold automaton with \( k \) guards is the concatenation of at most \( k+1 \) steady paths.

Using this proposition, we first construct a formula \( \phi_{\text{steady}} \) such that \( \phi_{\text{steady}}(\sigma, \sigma') \) holds iff there is a steady path \( \text{path}(\sigma, \tau) \) such that \( \tau(\sigma) = \sigma' \).
The formula $\phi_{\text{steady}}$. For every rule $r \in \mathcal{R}$, let $x_r$ be a variable ranging over non-negative integers. Intuitively, the value of $x_r$ will represent the number of times $r$ is fired during the (supposed) path from $\sigma$ to $\sigma'$. Let $X = \{x_r\}_{r \in \mathcal{R}}$. We construct $\phi_{\text{steady}}$ step by step, specifying necessary conditions for $\sigma, \sigma'$ and $X$ to satisfy the existence of the steady path, which in particular implies that $\sigma'$ is reachable from $\sigma$.

**Step 1.** $\sigma$ and $\sigma'$ must have the same values of the parameters, which must satisfy the resilience condition, the same number of processes, and the same context:

$$\phi_{\text{base}}(\sigma, \sigma') \equiv \sigma.p = \sigma'.p \land \text{RC}(\sigma.p) \land N(\sigma.p) = N(\sigma'.p) \land \omega(\sigma) = \omega(\sigma').$$

**Step 2.** For a location $\ell \in \mathcal{L}$, let $\text{out}^i_\ell, \ldots, \text{out}^a_\ell$ be all outgoing rules from $\ell$ and let $\text{in}^i_\ell, \ldots, \text{in}^b_\ell$ be all incoming rules to $\ell$. The number of processes in $\ell$ after the execution of the path is the initial number, plus the incoming processes, minus the outgoing processes. Since $x_r$ models the number of times the rule $r$ is fired, we have

$$\phi_{\ell}(\sigma, \sigma', X) \equiv \bigwedge_{\ell \in \mathcal{L}} \left( \sum_{i=1}^{a_\ell} x_{\text{in}^i_\ell} - \sum_{j=1}^{b_\ell} x_{\text{out}^j_\ell} = \sigma'.\kappa(\ell) - \sigma.\kappa(\ell) \right)$$

**Step 3.** Similarly, for the shared variables we must have:

$$\phi_{\Gamma}(\sigma, \sigma', X) \equiv \bigwedge_{z \in \Gamma} \left( \sum_{r \in \mathcal{R}} (x_r \cdot r.u[z]) = \sigma'.g[z] - \sigma.g[z] \right)$$

**Step 4.** Since $\text{path}(\sigma, \tau)$ must be steady, if a rule is fired along $\text{path}(\sigma, \tau)$ then its guard must be true in $\sigma$ and so

$$\phi_{\mathcal{R}}(\sigma, X) \equiv \bigwedge_{r \in \mathcal{R}} x_r > 0 \Rightarrow (\sigma \models r.\varphi)$$

**Step 5.** Finally, for every rule $r$ that occurs in $\text{path}(\sigma, \tau)$, the path must contain a “fireable” chain leading to $r$, i.e., a set of rules $S = \{r_1, \ldots, r_s\} \subseteq \mathcal{R}$ such that all rules of $S$ are executed in $\text{path}(\sigma, \tau)$, there is a process in $\sigma$ at $r_1.\text{from}$, and the rules $r_1, \ldots, r_s$ form a chain leading from $r_1.\text{from}$ to $r.\text{from}$. We capture this by the constraint

$$\phi_{\text{appl}}(\sigma, X) \equiv \bigwedge_{r \in \mathcal{R}} \left( x_r > 0 \Rightarrow \bigvee_{S = \{r_1, r_2, \ldots, r_s\} \subseteq \mathcal{R}} \phi^r_{\text{chain}}(S, \sigma, X) \right)$$

where

$$\phi^r_{\text{chain}}(S, \sigma, X) \equiv \bigwedge_{r \in S} x_r > 0 \land \sigma.\kappa(r_1.\text{from}) > 0 \land \bigwedge_{1 < i \leq s} r_{i-1}.\text{to} = r_i.\text{from} \land r_s = r$$
Combining the steps. Define $\phi_{\text{steady}}(\sigma, \sigma')$ as follows:

$$\phi_{\text{steady}}(\sigma, \sigma') \equiv \phi_{\text{base}}(\sigma, \sigma') \land \exists X \geq 0. \phi_L(\sigma, \sigma', X) \land \phi_R(\sigma, X) \land \phi_{\text{appl}}(\sigma, X).$$

where $\exists X \geq 0$ abbreviates $\exists x_{r_1} \geq 0, \ldots, \exists x_{r_{|R|}} \geq 0$. By our discussion, it is clear that if there is a steady path leading from $\sigma$ to $\sigma'$, then $\phi_{\text{steady}}(\sigma, \sigma')$ is satisfiable. The following theorem proves the converse.

**Theorem 4.** Let $\mathcal{T}A$ be a threshold automaton and let $\sigma, \sigma' \in \Sigma$ be two configurations. Formula $\phi_{\text{steady}}(\sigma, \sigma')$ is satisfiable if and only if there is a steady schedule $\tau$ with $\tau(\sigma) = \sigma'$.

Observe that, while $\phi_{\text{steady}}$ has exponential length in $\mathcal{T}A$ when constructed naively (because of the exponentially many disjunctions in $\phi_{\text{appl}}$), its satisfiability is in NP. Indeed, we first non-deterministically guess one of the disjunctions for each conjunction of $\phi_{\text{appl}}$ and then check in non-deterministic polynomial time that the (polynomial sized) formula with only these disjuncts is satisfiable. This is possible because existential Presburger arithmetic is known to be in NP [14].

The formula $\phi_{\text{reach}}$. By Proposition 1 every path from $\sigma$ to $\sigma'$ in a threshold automaton with a set $\Phi$ of guards can be written in the form

$$\sigma = \sigma_0 \rightarrow \sigma_0' \rightarrow \sigma_1 \rightarrow \sigma_1' \rightarrow \ldots \rightarrow \sigma_K \rightarrow \sigma_K' = \sigma'$$

where $K = |\Phi| + 1$, and $\sigma_i \rightarrow \sigma_i'$ is a steady path for each $0 \leq i \leq K$. It is easy to see from the definition of the transition relation between configurations that we can construct a polynomial sized existential Presburger formula $\phi_{\text{step}}$ such that $\phi_{\text{step}}(\sigma, \sigma')$ is true iff $\sigma'$ can be reached from $\sigma$ by firing at most one rule. Thus, we define $\phi_{\text{reach}}(\sigma, \sigma')$ to be

$$\exists \sigma_0, \sigma_0', \ldots, \sigma_K, \sigma_K' \left( \sigma_0 = \sigma \land \sigma_K' = \sigma' \land \bigwedge_{0 \leq i \leq K} \phi_{\text{steady}}(\sigma_i, \sigma_i') \land \bigwedge_{0 \leq i \leq K-1} \phi_{\text{step}}(\sigma_i', \sigma_{i+1}) \right)$$

**Theorem 5.** Given a threshold automaton $\mathcal{T}A$, there is an existential Presburger formula $\phi_{\text{reach}}$ such that $\phi_{\text{reach}}(\sigma, \sigma')$ holds iff $\sigma \rightarrow \sigma'$.

As deciding the truth of existential Presburger formulas is in NP, we obtain:

**Corollary 1.** The reachability and parameterized reachability problems are in NP.

**Remark 1.** In [15] an algorithm was given for parameterized reachability of threshold automata in which the updates of all rules contained in loops are equal to 0. Our algorithm does not need this restriction.
5 Safety and Liveness

We recall the definition of Fault-Tolerant Temporal Logic (ELTL\textsubscript{FT}), the fragment of LTL used in [19] to specify and verify properties of a large number of fault-tolerant algorithms. ELTL\textsubscript{FT} has the following syntax, where $S \subseteq \mathcal{L}$ is a set of locations and $\text{guard} \in \Phi$ is a guard:

$$
\begin{align*}
\psi &::= \text{pf} \mid G \psi \mid F \psi \mid \psi \land \psi \\
pf &::= \text{cf} \mid \text{gf} \Rightarrow \text{cf} \\
gf &::= \text{guard} \mid \text{gf} \land \text{gf} \mid \text{gf} \lor \text{gf}
\end{align*}
$$

An infinite path $\text{path}(\sigma, \tau)$ starting at $\sigma = (\kappa, g, p)$, satisfies $S = 0$ if $\kappa(\ell) = 0$ for every $\ell \in S$, and $\text{guard}$ if $(g, p)$ satisfies $\text{guard}$. The rest of the semantics is standard. The negations of specifications of the benchmarks [5,27,3,23,24,13,7,4,26] can be expressed in ELTL\textsubscript{FT}, as we are interested in finding possible violations.

Example 3. One specification of the algorithm from Figure 1 is that if $\text{myval}_i = 1$ for every correct process $i$, then eventually $\text{accept}_j = \text{true}$ for some correct process $j$. In the words of the automaton from Figure 2, a violation of this property would mean that initially all correct processes are in location $\ell_0$, but no correct process ever reaches location $\ell_3$. In ELTL\textsubscript{FT} we write this as\footnote{\texttt{\{\ell_0,\ell_2,\ell_3\} = 0 \land G \{\ell_3\} = 0}.}\

This has to hold under the fairness constraint

$$
G F \left( ((x \geq t + 1 \lor x \geq n - t) \Rightarrow \{\ell_0\} = 0) \land \{\ell_1\} = 0 \land (x \geq n - t \Rightarrow \{\ell_2\} = 0) \right).
$$

As we have self-loops at locations $\ell_0$ and $\ell_2$, a process could stay forever in one of these two states, even if it has collected enough messages, i.e., if $x \geq t + 1$ or $x \geq n - t$. This is the behavior that we want to prevent with such a fairness constraint. Enough sent messages should force each process to progress, so the location eventually becomes empty. Similarly, as the rule leading from $\ell_1$ has a trivial guard, we want to make sure that all processes starting in $\ell_1$ eventually (send a message and) leave $\ell_1$ empty, as required by the algorithm.

In this section we study the following problem:

Definition 4 (Parameterized safety and liveness). Given a threshold automaton $\mathcal{T}A$ and a formula $\varphi$ in ELTL\textsubscript{FT}, check whether there is an initial configuration $\sigma_0$ and an infinite schedule $\tau$ applicable to $\sigma_0$ such that $\text{path}(\sigma_0, \tau) \models \varphi$.

Since parameterized coverability is NP-hard, it follows that parameterized safety and liveness is also NP-hard. We prove that for automata with multiplicative environments (see Definition 2) parameterized safety and liveness is in NP.

Theorem 6. Parameterized safety and liveness of threshold automata with multiplicative environments is in NP.
The proof, which can be found in the Appendix, is very technical, and we only give a rough sketch here. The proof relies on two notions introduced in [19].

First, it is shown in [19] that every ELTL formula is equivalent to a formula in normal form of shape \( \phi_0 \land F \phi_1 \land \cdots \land F \phi_k \land G \phi_{k+1} \), where \( \phi_0 \) is a propositional formula and \( \phi_1, \ldots, \phi_{k+1} \) are themselves in normal form. Further, formulas can be put in normal form in polynomial time. The second notion introduced in [19] is the cut graph \( Gr(\phi) \) of a formula in normal form. For our sketch it suffices to know that \( Gr(\phi) \) is a directed acyclic graph with two special nodes \( loop_{st} \) and \( loop_{end} \), and every other node is a subformula of \( \varphi \) in normal form (see Figure 4).

For a formula \( \varphi \equiv \phi_0 \land F \phi_1 \land \cdots \land F \phi_k \land G \phi_{k+1} \), we will say that its local proposition is \( \phi_0 \) and its global proposition is the local proposition of \( \phi_{k+1} \). It is shown in [19] that, given \( \varphi = \phi_0 \land F \phi_1 \land \cdots \land F \phi_k \land G \phi_{k+1} \), some infinite path satisfies \( \varphi \) iff there exists a topological ordering \( v_0, v_1, \ldots, v_c = loop_{st}, v_c+1, \ldots, v_l = loop_{end} \) of the cut graph and a path \( \sigma_0, \tau_0, \sigma_1, \ldots, \sigma_c, \tau_c, \ldots, \sigma_{l-1}, \tau_{l-1}, \sigma_l \) such that, roughly speaking, (among other technical conditions) every configuration \( \sigma_i \) satisfies the local proposition of \( v_i \) and every configuration in \( Cfgs(\sigma_i, \tau_i) \) satisfies the global proposition of every \( v_j \) where \( j \leq i \).

Using multiplicativity and our main result that reachability is definable in existential Presburger arithmetic, we show that for every proposition \( p \), we can construct an existential formula \( \phi_p(\sigma, \sigma') \) such that: If there is a path between \( \sigma \) and \( \sigma' \), all of whose configurations satisfy \( p \), then \( \phi_p(\sigma, \sigma') \) is satisfiable. Further, if \( \phi_p(\sigma, \sigma') \) is satisfiable, then there is a path between \( 2 \cdot \sigma \) and \( 2 \cdot \sigma' \) all of whose configurations satisfy \( p \). (Here \( 2 \cdot \sigma = ((2 \cdot \sigma), (2 \cdot \sigma), (2 \cdot \sigma)) \)). Then, once we have fixed a topological ordering \( V = v_0, \ldots, v_l \), (among other conditions), we check if there are configurations \( \sigma_0, \ldots, \sigma_l \) such that for every \( i, \sigma_i \) satisfies the local proposition of \( v_i \) and for every \( j \leq i \), \( \phi_p(\sigma_i, \sigma_{i+1}) \) is satisfiable where \( \sigma_l \) is the global proposition of \( v_l \). Using multiplicativity, we then show that this procedure is sufficient to check if the given specification \( \varphi \) is satisfied.

Our algorithm consists therefore of the following steps: (1) bring \( \varphi \) in normal form; (2) construct the cut graph \( Gr(\varphi) \); (3) guess a topological ordering of the nodes of \( Gr(\varphi) \); (4) for the guessed ordering, check in nondeterministic polynomial time if the required sequence \( \sigma_0, \ldots, \sigma_l \) exists.

Remark 2. From an algorithm given in [19] one can infer that parameterized safety and liveness is in NP for threshold automata with multiplicative environments, where all cycles are simple, and rules in cycles have update \( 0 \). (The NP bound was not explicitly given in [19].) Our algorithm only requires multiplicativity.
6 Synthesis of Threshold Guards

We study the *bounded synthesis* problem for constructing parameterized threshold guards in threshold automata satisfying a given specification.

**Sketch threshold automata.** Let an *indeterminate* be a variable that can take values over rational numbers. We consider threshold automata whose guards can contain indeterminates. More precisely, a *sketch* threshold automaton is a tuple $TA = (L, I, G, R)$, just as before, except for the following change. Recall that in a threshold automaton, a guard is an inequality of one of the following two forms:

$$x \geq a_0 + a_1 \cdot p_1 + \ldots + a_{|G|} \cdot p_{|G|} \quad \text{or} \quad x < a_0 + a_1 \cdot p_1 + \ldots + a_{|G|} \cdot p_{|G|}$$

where $a_0, a_1, \ldots, a_{|G|}$ are rational numbers. In a sketch threshold automaton, some of the $a_0, a_1, \ldots, a_{|G|}$ can be *indeterminates*. Moreover, indeterminates can be shared between two or more guards.

Given a sketch threshold automaton $TA$ and an assignment $\mu$ to the indeterminates, we let $TA[\mu]$ denote the threshold automaton obtained by substituting the indeterminates by their values in $\mu$. We define the *bounded synthesis* problem:

Given: An environment $Env$, a sketch threshold automaton $TA$ with indeterminates $v_1, \ldots, v_m$, a formula $\phi$ of ELTL$_{RT}$, and a polynomial $p$.

Decide: Is there an assignment $\mu$ to $v_1, \ldots, v_m$ of size $O(p(|TA| + |\phi|))$ (i.e., the vector $(\mu(v_1), \ldots, \mu(v_m))$ of rational numbers can be encoded in binary using $O(p(|TA| + |\phi|))$ bits) such that $TA[\mu]$ satisfies $\neg \phi$ (i.e., such that for every initial configuration $\sigma_0$ in $TA[\mu]$, every infinite run starting from $\sigma_0$ satisfies $\neg \phi$)?

We say that an assignment $\mu$ to the indeterminates makes the environment multiplicative if the conditions of Definition 2 are satisfied after plugging in the assignment $\mu$ in the automaton. In the following, we will only be concerned with assignments which make the environment multiplicative.

Since we can guess an assignment in polynomial time, by Theorem 6 it follows

**Theorem 7.** *Bounded synthesis is in $\Sigma^p_2$.*

By a reduction from the $\Sigma_2$-SAT problem, we also provide a matching lower bound.

**Theorem 8.** *Bounded synthesis is $\Sigma^p_2$-complete.*

The *synthesis* problem is defined as the bounded synthesis problem, but lifting the constraint on the size of $\mu$. While we do not know the exact complexity of the synthesis problem, we can show that, for a large and practically motivated class of threshold automata introduced in [22], the synthesis problem reduces to the bounded synthesis problem. We proceed to describe and motivate the class.
The parameter variables of fault-tolerant distributed algorithms usually consist of a variable \( n \) denoting the number of processes running the algorithm and various “failure” variables for the number of processes exhibiting different kinds of failures (for example, a variable \( t_1 \) might be used to specify the number of Byzantine failures, a variable \( t_2 \) for crash failures, etc.). The following three observations are made in [22]:

1. The resilience condition of these algorithms is usually of the form \( n > \sum_{i=1}^{k} \delta_i t_i \) where \( t_i \) are parameter variables and \( \delta_i \) are natural numbers.
2. Threshold guards typically serve one of two purposes: to check if at least a certain fraction of the processes sends a message (for example, \( x > n/2 \) ensures that a strict majority of processes has sent a message), or to bound the number of processes that crash.
3. The coefficients of the guards are rational numbers with small denominators (typically at most 3).

By (2), the structure of the algorithm guarantees that the value of a variable \( x \) never goes beyond \( n \), the number of processes. Therefore, given a threshold guard template \( x \bowtie u \cdot \pi + v \), where \( u \) is a vector of indeterminates, \( \pi \) is a vector of parameter variables, \( v \) is an indeterminate, and \( \bowtie \) is either \( \geq \) or \( < \), we are only interested in assignments \( \mu \) of \( u \) and \( v \) which satisfy \( 0 \leq \mu(u) \cdot \nu(\pi) + \mu(v) \leq n \) for every valuation \( \nu(\pi) \) of \( \pi \) respecting the resilience condition. Guards obtained by instantiating guard templates with such a valuation \( \mu \) are called sane guards [22].

The following result is proved in [22]: Given a resilience condition \( n > \sum_{i=1}^{k} \delta_i t_i \), and an upper bound \( D \) on the denominator of the entries of \( \mu \) (see (1) and (3) above), the numerators of the entries of \( \mu \) are necessarily of polynomial size in \( k, \delta_1, \ldots, \delta_k \). Therefore, the synthesis problem for sane guards and bounded denominator, as introduced in [22], reduces to the bounded synthesis problem, and so it can be solved in \( \Sigma_2^p \) time. Moreover, the reduction used in Theorem 8 to prove \( \Sigma_2^p \)-hardness yields sketch threshold automata with sane guards, and so the synthesis problem for sane guards and bounded denominator is also \( \Sigma_2^p \)-complete.

7 Experimental Evaluation

Following the techniques presented in this paper, we have verified a number of threshold-based fault-tolerant distributed algorithms.

**Benchmarks.** Consistent broadcast (strb) [27] is given in Figure 1 and its threshold automaton is depicted in Figure 2. The algorithm is correct if in any execution either all correct processes or none set accept to true; moreover, if all correct processes start with value 0 then none of them accept, and if all correct processes start with value 1 then they all accept. The algorithm is designed to tolerate Byzantine failures of less than one third of processes, that is, if \( n > 3t \). Folklore Reliable Broadcast (frb) [5] that tolerates crash faults and Asynchronous Byzantine agreement (aba) [3] satisfy the same specifications as consistent broadcast, under the same resilience condition.
Table 1. The experiments were run on a machine with Intel® Core™ i5-7200U CPU with 7.7 GiB memory. The time limit was set to be 5 hours and the memory limit was set to be 7 GiB. TLE (MLE) means that the time limit (memory limit) exceeded for the particular benchmark.

Non-blocking atomic commit (nbacr) [24] and (nbacg) [13] deal with faults using failure detectors. We model this by introducing a special location such that a process is in it if and only if it suspects that there is a failure of the system.

Condition-based consensus (cbc) [23] reaches consensus under the condition that the difference between the numbers of processes initialized with 0 and 1 differ by at least t, an upper bound on the number of faults. We also check algorithms that allow consensus to be achieved in one communication step, such as cfcs [7], c1cs [4], as well as Byzantine One Step Consensus bosco [26].

Evaluation. Table 1 summarizes our results and compares them with the results obtained using the ByMC tool [18]. Due to lack of space, we have omitted those experiments for which both ByMC and our tool took less than 10 seconds.
We implemented our algorithms in Python and used Z3 as a back-end SMT solver for solving the constraints over existential Presburger arithmetic. Our implementation takes as input a threshold automaton and a specification in ELTL_{FT} and checks if a counterexample exists. We apply to the latest version of the benchmarks of [18]. Each benchmark yields two threshold automata, a hand-coded one and one obtained by a data abstraction of the algorithm written in Parametric Promela. For automata of the latter kind, due to data abstraction, we have to consider different cases for the same algorithm. We test each automaton against all specifications for that automaton.

Our tool outperforms ByMC in all automata with more than 30 states, with the exception of the second case of cbc. It performs worse in most small cases, however in these cases, both ByMC and our tool take less than 10 seconds. ByMC works by enumerating all so-called schemas of a threshold automaton, and solving a SMT problem for each of them; the number of schemas can grow exponentially in the number of guards. Our tool avoids the enumeration. Since the number of schemas for the second case of cbc is just 2, while the second case of aba and third case of cf1s have more than 3000, avoiding the enumeration seems to be key to our better performance.

8 Conclusions

In this paper we have addressed the complexity of the most important verification and synthesis problems for threshold automata. In particular, we have shown that the coverability and reachability problems, as well as the model checking problem for the fault-tolerant temporal logic ELTL_{FT} are all NP-complete, and that the bounded synthesis problem is \( \Sigma_{2}^{p} \)-complete. These results are a consequence of a novel characterization of the reachability relation of threshold automata as an existential formula of Presburger arithmetic.

References

1. Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder. Decidability of Parameterized Verification. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015.
2. Michael Blondin, Christoph Haase, and Filip Mazowiecki. Affine extensions of integer vector addition systems with states. In CONCUR, volume 118 of LIPIcs, pages 14:1–14:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.
3. Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. J. ACM, 32(4):824–840, 1985.
4. Francisco Vilar Brasileiro, Fabiola Greve, Achour Mostefaoui, and Michel Raynal. Consensus in one communication step. In PaCT, volume 2127 of LNCS, pages 42–50, 2001.
5. Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. J. ACM, 43(2):225–267, 1996.
6. Giorgio Delzanno, Arnaud Sangnier, Riccardo Traverso, and Gianluigi Zavattaro. On the complexity of parameterized reachability in reconfigurable broadcast networks. In FSTTCS 2012, pages 289–300, 2012.
7. Dan Dobre and Neeraj Suri. One-step consensus with zero-degradation. In *DSN*, pages 137–146, 2006.
8. Catherine Dufourd, Alain Finkel, and Philippe Schnoebelen. Reset nets between decidability and undecidability. In *ICALP*, volume 1443 of *Lecture Notes in Computer Science*, pages 103–115. Springer, 1998.
9. Javier Esparza. Decidability and complexity of petri net problems - An introduction. In *Petri Nets*, volume 1491 of *LNCS*, pages 374–428. Springer, 1996.
10. Javier Esparza, Alain Finkel, and Richard Mayr. On the verification of broadcast protocols. In *LICS*, pages 352–359. IEEE Computer Society, 1999.
11. Javier Esparza and Mogens Nielsen. Decidability issues for petri nets - A survey. *Bulletin of the EATCS*, 52:244–262, 1994.
12. Steven M. German and A. Prasad Sistla. Reasoning about systems with many processes. *J. ACM*, 39(3):675–735, 1992.
13. Rachid Guerraoui. Non-blocking atomic commit in asynchronous distributed systems with failure detectors. *Distributed Computing*, 15(1):17–25, 2002.
14. Christoph Haase. A survival guide to presburger arithmetic. *ACM SIGLOG News*, 5(3):67–82, 2018.
15. Igor Konnov, Marijana Lazic, Helmut Veith, and Josef Widder. Para$^2$: parameterized path reduction, acceleration, and SMT for reachability in threshold-guarded distributed algorithms. *Formal Methods in System Design*, 51(2):374–428, 2017.
16. Igor Konnov, Helmut Veith, and Josef Widder. On the completeness of bounded model checking for threshold-based distributed algorithms: Reachability. In *CONCUR*, volume 8704 of *LNCS*, pages 125–140, 2014.
17. Igor Konnov, Helmut Veith, and Josef Widder. On the completeness of bounded model checking for threshold-based distributed algorithms: Reachability. *Information and Computation*, 252:95–109, 2017.
18. Igor Konnov and Josef Widder. Bymc: Byzantine model checker. In *ISoLA (3)*, volume 11246 of *LNCS*, pages 327–342. Springer, 2018.
19. Igor V. Konnov, Marijana Lazic, Helmut Veith, and Josef Widder. A short counterexample property for safety and liveness verification of fault-tolerant distributed algorithms. In *POPL 2017*, pages 719–734, 2017.
20. Jure Kukovec, Igor Konnov, and Josef Widder. Reachability in parameterized systems: All flavors of threshold automata. In *CONCUR*, pages 19:1–19:17, 2018.
21. Richard E. Ladner. The circuit value problem is log space complete for p. *SIGACT News*, 7(1):1820, 1975.
22. Marijana Lazić, Igor Konnov, Josef Widder, and Roderick Bloem. Synthesis of distributed algorithms with parameterized threshold guards. In *OPODIS*, volume 95 of *LIPIcs*, pages 32:1–32:20, 2017.
23. Achour Mostefaioui, Eric Mourgaya, Philippe Raipin Parvédy, and Michel Raynal. Evaluating the condition-based approach to solve consensus. In *DSN*, pages 541–550, 2003.
24. Michel Raynal. A case study of agreement problems in distributed systems: Non-blocking atomic commitment. In *HASE*, pages 209–214, 1997.
25. Sylvain Schmitz and Philippe Schnoebelen. The power of well-structured systems. *CoRR*, abs/1402.2908, 2014.
26. Yee Jiun Song and Robbert van Renesse. Bosco: One-step Byzantine asynchronous consensus. In *DISC*, volume 5218 of *LNCS*, pages 438–450, 2008.
27. T.K. Srikanth and Sam Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Dist. Comp.*, 2:80–94, 1987.
APPENDIX

A Detailed proofs from Section 3

The intuition behind the proof of theorem 1 has been discussed in section 3. Here we provide the formal proof of the theorem.

The reduction for theorem 2 is given by means of the following definition, since the same construction will be used later on.

Definition 5. Given a 3-CNF formula \( \varphi = C_1 \land C_2 \land \cdots \land C_m \) over variables \( x_1, \ldots, x_n \), we define its threshold automaton \( TA_\varphi = (L, I, V, R) \) and the associated environment \( Env_\varphi \) as follows: (see an illustrative example in Figure 3)

- \( Env_\varphi = (II, RC, N) \) where II consists of a single parameter \( k \), RC is simply true and \( N(k) = k \).
- The set of locations is \( L = \{ \ell_1, \ldots, \ell_n, T_1, \ldots, T_n, \bot_1, \ldots, \bot_n, \ell_{\text{mid}}, \ell_F \} \).
- The set of initial locations is \( I = \{ \ell_1, \ldots, \ell_n \} \).
- The variable set contains shared variables \( V = \{ y_1, \ldots, y_n, \bar{y}_1, \ldots, \bar{y}_n, c_1, \ldots, c_m \} \).
- The set of rules \( R \) contains:
  - For every \( 1 \leq i \leq n \), we have \( (\ell_i, \top_i, \bar{y}_i < 1, y_i++) \in R \) and \( (\ell_i, \bot_i, y_i < 1, \bar{y}_i++) \in R \).
  - For every \( 1 \leq i \leq n \), we have \( (\top_i, \ell_{\text{mid}}, \text{true}, c_{j_1}++, \ldots, c_{j_s}++) \in R \) where \( C_{j_1}, \ldots, C_{j_s} \) are all the clauses that \( x_i \) appears in.
  - For every \( 1 \leq i \leq n \), we have \( (\bot_i, \ell_{\text{mid}}, \text{true}, c_{j_1}++, \ldots, c_{j_s}++) \in R \) where \( C_{j_1}, \ldots, C_{j_s} \) are all the clauses that \( \neg x_i \) appears in.
  - \( (\ell_{\text{mid}}, \ell_F, \phi, 0) \in R \), where \( \phi = c_1 \geq 1 \land \ldots \land c_m \geq 1 \).

Remark: Note that the constructed threshold automaton is acyclic and has only constant guards, i.e., only guards of the form \( x \top \) or \( x \bot \). For every \( 1 \leq i \leq n \), \( \ell_i \top_i \) \( \bot_i \) no process can move to \( \top_i \) (resp. \( \bot_i \)).

Lemma 1. Let \( \varphi \) be a 3-CNF formula over variables \( x_1, \ldots, x_n \), and let its corresponding threshold automaton be \( TA_\varphi \). If \( \sigma_0 \) is an initial configuration and \( \tau \) is a schedule applicable to \( \sigma_0 \) then the following holds for every \( i \in \{ 1, \ldots, n \} : \) If there exists \( \sigma \in \text{Cfgs}(\sigma_0, \tau) \) with \( \sigma.\kappa[\top_i] \geq 1 \) (resp. \( \sigma.\kappa[\bot_i] \geq 1 \)), then for every \( \sigma' \in \text{Cfgs}(\sigma_0, \tau) \) it holds that \( \sigma'.\kappa[\bot_i] = 0 \) (resp. \( \sigma'.\kappa[\top_i] = 0 \)).

Proof. Let \( \text{path}(\sigma_0, \tau) = \sigma_0, t_1, \sigma_1, \ldots, \sigma_{|\tau|-1}, t_{|\tau|}, \sigma_{|\tau|} \). Let \( i \in \{ 1, \ldots, n \} \) and let \( \sigma_p \) be the first configuration in the path such that \( \sigma_p.\kappa[\top_i] \geq 1 \). Hence \( t_{p-1} = (\ell_i, \top_i, \bar{y}_i < 1, y_i++) \). This means that \( \sigma_{p-1}.g[\bar{y}_i] = 0 \) and so the rule \( (\ell_i, \bot_i, y_i < 1, \bar{y}_i++) \) could not have been fired before \( \sigma_{p-1} \). Since this rule was never fired before \( \sigma_{p-1} \) it follows that \( \sigma_q.\kappa[\bot_i] = 0 \) for all \( q \leq p \). Also, notice that \( \sigma_p.g[\bar{y}_i] \geq 1 \) and so the rule \( (\ell_i, \bot_i, y_i < 1, \bar{y}_i++) \) cannot be fired after \( \sigma_p \). Consequently we get that \( \sigma_q.\kappa[\bot_i] = 0 \) for all \( q > p \). The other claim is proven in a similar manner as well.

17
A.1 Proof of Theorem 1

Proof. We reduce from 3-SAT. Let \( \varphi = C_1 \land C_2 \land \cdots \land C_m \) be a 3-CNF formula over variables \( x_1, \ldots, x_n \) and consider the threshold automaton \( TA_\varphi \) and the environment \( Env_\varphi \) as described in Definition 5. We show that \( \varphi \) is satisfiable if and only if there is a path \( \text{path}(\sigma_0, \tau) \) such that the configuration \( \tau(\sigma_0) \) covers \( \ell_F \).

(⇒) Suppose \( \varphi \) is satisfiable and \( \upsilon \) is a truth assignment that satisfies \( \varphi \). We define an initial configuration \( \sigma_0 \) and a schedule \( \tau \) as follows.

The initial configuration \( \sigma_0 \) has one process in each of the initial locations \( \ell_1, \ldots, \ell_n \). As usual, all shared variables initially have value 0.

The schedule \( \tau \) is defined as follows. For each \( i \in \{1, \ldots, n\} \), if \( \upsilon(x_i) = \text{true} \) we fire the rule \( (\ell_i, \top, \bar{y}_i < 1, \bar{y}_i++) \), and if \( \upsilon(x_i) = \text{false} \) then we fire the rule \( (\ell_i, \bot, \bar{y}_i < 1, \bar{y}_i++) \). As all these guards initially evaluate to true, these \( n \) rules are applicable to \( \sigma_0 \) (in any order), and lead to the (same) configuration \( \sigma_1 \) such that \( \sigma_1, k[\top_i] \geq 1 \) if and only if \( \upsilon(x_i) = \text{true} \) and \( \sigma_1, k[\bot_i] \geq 1 \) if and only if \( \upsilon(x_i) = \text{false} \), and all the other locations are empty. Note that by construction, for each non-empty location from \( \sigma_1 \), there is exactly one outgoing rule. Therefore, the rest of the schedule is formed in the only possible way, that is, each process moves to \( \ell_{\text{mid}} \) and increments the corresponding shared variable(s) along that rule.

Since each clause \( C_j \) evaluates to true under the assignment \( \upsilon \), every shared variable \( c_j \) is incremented at least once. Hence the guard of the rule leading to \( \ell_F \) becomes true, and so each process can move to \( \ell_F \), thereby covering \( \ell_F \).

(⇐) Suppose there is an initial configuration \( \sigma_0 \) and a schedule \( \tau \) such that \( \tau(\sigma_0) = \sigma \) and \( \sigma, k[\ell_F] \geq 1 \). We construct the truth assignment \( \upsilon \) as follows: for each \( i \in \{1, \ldots, n\} \) we define \( \upsilon(x_i) = \text{true} \) if \( \top_i \) was visited along the path, i.e., if there is \( \sigma' \in \text{Cfgs}(\sigma_0, \tau) \) with \( \sigma', k[\top_i] \geq 1 \); similarly we define \( \upsilon(x_i) = \text{false} \), if there is \( \sigma' \in \text{Cfgs}(\sigma_0, \tau) \) with \( \sigma', k[\bot_i] \geq 1 \). If there is an \( i \) such that neither \( \top_i \) nor \( \bot_i \) was visited by \( \text{path}(\sigma_0, \tau) \), then we define \( \upsilon(x_i) = \text{true} \). Lemma 1 implies that \( \upsilon \) is a well-defined truth assignment. By construction, it is easy to see that \( \upsilon \) satisfies \( \varphi \).

A.2 Proof of Theorem 2

Proof. Once again we give a reduction from 3-SAT. Let \( \varphi \) be a 3-CNF formula over variables \( x_1, \ldots, x_n \). We consider the threshold automaton \( TA_\varphi \) as described in Definition 5 and remove all the guards and updates on any rule that leaves any initial location. Formally, for every \( i \in \{1, \ldots, n\} \), we replace the rule \( (\ell_i, \top, \bar{y}_i < 1, \bar{y}_i++) \) with \( (\ell_i, \top, \text{true}, \text{false}) \). Similarly, we replace \( (\ell_i, \bot, \bar{y}_i < 1, \bar{y}_i++) \) with \( (\ell_i, \bot, \text{true}, \text{false}) \). The rest of the rules in \( TA_\varphi \) are retained as they are. We will denote this new threshold automaton by \( TA'_\varphi \).

We define the initial configuration \( \sigma_0 \) such that there is exactly one process in each initial state \( \ell_i \) (as usual, all other locations are empty and all shared variables have values 0).
\(\Rightarrow\) Suppose \(\varphi\) can be satisfied with assignment \(v\). Notice that in this case, we already described a path from \(\sigma_0\) covering \(\ell_F\) in the previous proof.

\(\Leftarrow\) Suppose \(\ell_F\) can be covered from \(\sigma_0\). Let \(\text{path}(\sigma_0, \tau)\) be a path along which \(\ell_F\) is covered. Define \(v\) to be the following truth assignment for \(\varphi\): \(v(x_i) = \text{true}\) if the rule \((\ell_i, 1, \text{true, } 0)\) was fired in the schedule \(\tau\), \(v(x_i) = \text{false}\) if the rule \((\ell_i, 0, \text{true, } 0)\) was fired in the schedule \(\tau\), and \(v(x_i) = \text{true}\) if neither of these rules were fired. Since the automaton is acyclic and there is only one process at \(\ell_i\), both \((\ell_i, 1, \text{true, } 0)\) and \((\ell_i, 0, \text{true, } 0)\) cannot be fired and so this is a well-defined assignment. It can then be easily verified that \(v\) is a satisfying assignment for \(\varphi\).

### A.3 Proof of Theorem 3

**Proof.** For proving P-hardness, we give a logspace reduction from the Circuit Value Problem (CVP) which is known to be P-complete [21]. The reduction we present is similar to the reduction given in Proposition 1 of [6].

CVP is defined as follows: Given a boolean circuit \(C\) with \(n\) input variables and \(m\) gates, and a truth assignment \(v\) for the input variables, check if \(v\) evaluates to 1 on the circuit \(C\).

We represent each binary gate \(g\) as a tuple \((o, s_1, s_2)\), where \(o \in \{\wedge, \vee\}\) denotes the operation of \(g\), and \(s_1, s_2 \in \{x_1, x_2, ... , x_n, g_1, g_2, ..., g_m\}\) are the inputs to \(g\). In a similar fashion, each unary gate \(g\) is represented as a tuple \((\neg, s)\). By convention, \(g_1\) is always the output gate.

Based on this we present the following reduction: (See Figure 3 for an illustative example). Let \(C\) be a boolean circuit with input variables \(x_1, x_2, ..., x_n\) and gates \(g_1, g_2, ..., g_m\), and let \(v\) be a truth assignment that for every \(x_i\), assigns \(v(x_i) = b_i \in \{0, 1\}\). We define the threshold automaton \(\text{TA}_{C,v} = (\mathcal{L}, \mathcal{I}, \mathcal{V}, \mathcal{R})\) and the environment \(\text{Env}_{C,v} = (\Pi, RC, N)\) as follows:

- \(\text{Env}_{C,v} = (\Pi, RC, N)\) where \(\Pi\) consists of a single parameter \(k\), \(RC\) is simply true and \(N(k) = k\).
- The set of locations is \(\mathcal{L} = \{\ell_0, \ell_1, ..., \ell_n, \ell_{g_1}, \ell_{g_2}, ..., \ell_{g_m}, \ell_F\}\).
- The set of initial locations is \(\mathcal{I} = \{\ell_0, \ell_{g_1}, \ell_{g_2}, ..., \ell_{g_m}\}\).
- The set of shared variables is \(\mathcal{V} = \{x_1^0, x_1^1, x_2^0, x_2^1, ..., x_n^0, x_n^1, g_1^0, g_1^1, ..., g_m^0, g_m^1\}\).
- The set of rules \(\mathcal{R}\) contains:
  - a rule \((\ell_{i-1}, \ell_i, \text{true, } x_i^{b_i}+, +)\), for every \(1 \leq i \leq n\)
  - a rule \((\ell_{g_i}, \ell_F, g_i^1 \geq 1, 0)\), where \(g_i\) is the output gate,
  - rules \((\ell_g, \ell_{g_i}, s_i^{b_i} \geq 1 \land s_i^{b_2} \geq 1, g_i^{b_1b_2}++)\), for every binary gate \(g = (o, s_1, s_2)\), and for every \((b_1, b_2) \in \{0, 1\}^2\),
  - rules \((\ell_g, \ell_{g_i}, s_b \geq 1, g_i^{b}++)\), for every unary gate \(g = (\neg, s)\), and for every \(b \in \{0, 1\}\).

It can be easily checked that if \(\sigma_0\) is an initial configuration and \(\tau\) is a schedule such that \(\tau(\sigma_0) = \sigma\) then \(\sigma.g[x_i^{b_i}] \geq 1\) iff \(b = b_i\) and \(\sigma.g[g_i^b] \geq 1\) iff the gate \(g\) outputs \(b\) when the input variables are given the assignment \(v\). Consequently, the state \(\ell_F\) can be covered if and only if the circuit evaluates to 1 on the given assignment. Since the reduction can be clearly accomplished in logspace it follows that the coverability problem for this case is P-hard.
configuration \((c, \kappa)\) which is multiplicative (See definition 2). Clearly a guard of the form \(x \geq 0\) is redundant and so we assume that if \(c > 0\), \(g \in g_{\ell_1}^{k_{\ell_1} \cdot c} \) for any configuration \(\sigma\).

\(r\) is enabled at \(\sigma \implies r\) is enabled at \(\sigma + \sigma'\) for any configuration \(\sigma'\) \hspace{1cm} (1)

We prove that coverability in this case is in P by means of a saturation algorithm which begins with the set of initial locations and finishes with the set of all possible coverable locations. This algorithm is similar to the one mentioned in [4].

We initialize two variables \(X_L\) and \(X_R\) by \(X_L := I\) and \(X_R := \emptyset\), and repeatedly update them until a fixed point is reached, as follows:

- If there exists a rule \(r = (\ell, \ell', \text{true}, u) \in R \setminus X_R\) such that \(\ell \in X_L\), then set \(X_L := X_L \cup \{\ell'\}\) and \(X_R := X_R \cup \{r\}\).
- If there exists a rule \(r = (\ell, \ell', (\wedge_{1 \leq i \leq q} x_i \geq c_i), u) \in R \setminus X_R\) such that \(\ell \in X_L\), and there exists rules \(r_1, r_2, \ldots, r_q\) such that each \(r_i = (\ell_i, \ell_i', \varphi_i, u_i) \in X_R\) and \(u_i[x_i] > 0\), then set \(X_L := X_L \cup \{\ell'\}\) and \(X_R := X_R \cup \{r\}\).
By means of this algorithm we get a sequence of sets $X^n_L, X^n_R \subseteq (X^1_L, X^1_R) \subseteq \cdots \subseteq (X^n_L, X^n_R)$, one for each iteration. We say that a rule $r \in R$ can possibly occur if there is some initial configuration $\sigma_0$ such that $\sigma_0 \not\to \sigma$ and $\sigma$ enables $r$. We will now show that $X^n_R$ contains exactly the set of coverable locations and $X^n_L$ contains exactly the set of rules which can possibly occur.

First we show by induction on $i$, that if $\ell \in X^i_L$ then $\ell$ is coverable and if $r \in X^i_R$ then $r$ can possibly occur. The claim is clearly true for the base case of $i = 0$. Suppose the claim is true for some $i$. Let $X^i_L \setminus X^{i+1}_L = \{\ell'\}$ and $X^i_R \setminus X^{i+1}_R = \{r\}$. Let $c$ be the maximum constant appearing in any of the guards of any of the rules of TA.

By construction, there are two possible cases:

- $r$ is of the form $r = (\ell, \ell', true, u)$ such that $\ell \in X^i_L$. By inductive hypothesis $\ell$ is coverable. It is then clear that $\ell'$ is coverable and also that $r$ can possibly occur.

- $r$ is of the form $r = (\ell, \ell', (\land_{1 \leq j \leq q} x_j \geq c_j), u)$ such that $\ell \in X^i_L$. We have to show that $\ell'$ is coverable and $r$ can possibly occur.

By construction of the algorithm, for each $j \in \{1, \ldots, q\}$, there exists $r_j = (\ell_j, \ell'_j, \varphi_j, u_j) \in X^i_R$ such that $u_j[x_j] > 0$. By inductive hypothesis, $\ell$ is coverable and so there exists $\sigma^\ell_0 \not\to \sigma^\ell$ such that $\sigma, \varphi[\ell] \geq 1$. Once again, by inductive hypothesis, for each $j$, the rule $r_j$ can possibly occur and so there exists $\sigma^\ell_j \not\to \sigma^j$ such that $\sigma^j$ enables $r_j$.

Let $\sigma_0 = c \cdot (\sigma^\ell_0 + \sigma^\ell_1 + \cdots + \sigma^\ell_q)$ and let $\sigma_{mid} = (c \cdot (\sigma^\ell + \sigma^1 + \cdots + \sigma^q))$. Since the environment is multiplicative, $\sigma_0$ is a valid initial configuration. It is then easy to see (by repeated applications of observation 1) that $\sigma_0 \not\to \sigma_{mid}$. Notice that, by observation 1 $r_1, \ldots, r_q$ are all enabled at $\sigma_{mid}$ and also that $\sigma_{mid}[\ell_1] \geq c, \ldots, \sigma_{mid}[\ell_q] \geq c$. By firing each of $r_1, \ldots, r_q$ exactly $c$ many times, we can arrive at a configuration $\sigma$ where $r$ is enabled. Now firing $r$ from $\sigma$ covers $\ell'$. Hence $\ell'$ is coverable and $r$ can possibly occur thereby proving (2).

For the converse direction, suppose $\sigma_0$ is an initial configuration such that $\sigma_0 \not\to \sigma$ and $\sigma$ covers some location $\ell$ and enables some rule $r$. By an easy induction on the length of the path between $\sigma_0$ and $\sigma$ we can establish that $\ell \in X^i_L$ and $r \in X^i_R$.

**B Detailed proofs from Section 4**

In the following, when we say that $\phi_{steady}(\sigma, \sigma')$ holds with assignment $\{y_r\}_{r \in R}$ (where each $y_r$ is a natural number), we mean that the formula $\phi(\sigma, \sigma')$ is true when each variable $x_r$ is assigned the value $y_r$. Given two assignments $Y = \{y_r\}_{r \in R}$ and $Z = \{z_r\}_{r \in R}$ we say that $Z < Y$ iff $\sum_{r \in R} z_r < \sum_{r \in R} y_r$. Further for an assignment $Y = \{y_r\}_{r \in R}$ and a location $\ell$ we say that there is a fireable
cycle at \( \ell \) with respect to \( Y \) iff there exists rules \( T = \{t_1, \ldots, t_m\} \) such that \( y_i > 0, \ldots, y_m > 0, t_i.to = t_{i+1}.from \) for all \( i < m \), and \( t_m.to = t_1.from = \ell \).

The following two lemmas are important properties of the formula \( \phi_{steady} \).

**Lemma 2.** Let \( \phi_{steady}(\sigma, \sigma') \) be true with the assignment \( Y = \{y_r\}_{r \in R} \). Suppose \( y_i > 0 \) and \( \sigma[t]\text{from} > 0 \) for some rule \( t \). If there are no fireable cycles at \( t \) with respect to \( Y \), then \( \phi_{steady}(t(\sigma), \sigma') \) is true with an assignment \( Z < Y \).

**Lemma 3.** Let \( \phi_{steady}(\sigma, \sigma') \) be true with the assignment \( Y = \{y_r\}_{r \in R} \). Suppose \( \sigma[\ell] > 0 \) and suppose there is a fireable cycle \( \{t_1, \ldots, t_m\} \) at \( \ell \) with respect to \( Y \). Then \( \phi_{steady}(t_1(\sigma), \sigma') \) is true with an assignment \( Z < Y \).

We will first see how using these two lemmas, we can prove Theorem 4. Then we will present the proofs of these two lemmas.

**B.1 Proof of Theorem 4**

**Proof.** It is clear from our discussion during the construction of the formula \( \phi_{steady} \) that if there is a steady run from \( \sigma \) to \( \sigma' \) then \( \phi_{steady}(\sigma, \sigma') \) is true. Hence, we will only show that this is a sufficient condition.

Let \( \phi_{steady}(\sigma, \sigma') \) be true with the assignment \( \{y_r\}_{r \in R} \). We induct on the value of \( \sum_{r \in R} y_r \).

**Base Case.** It can be easily verified that \( \forall r \in R, y_r = 0 \) if and only if \( \sigma = \sigma' \). Hence, if all \( y_r \) are zero, then \( \sigma = \sigma' \) and so we are done. This constitutes the base case of the induction.

**Induction Hypothesis.** Let us fix values \( Y = \{y_r\}_{r \in R} \), such that not all of them are zero. We assume that if \( \phi_{steady}(c, c') \) holds for some configurations \( c \) and \( c' \) with the assignment \( Z = \{z_r\}_{r \in R} \) such that \( Z < Y \), then there is a steady schedule \( \tau \) such that \( \tau(c) = c' \).

**Induction Step.** Assume now that \( \phi_{steady}(\sigma, \sigma') \) holds with the assignment \( Y = \{y_r\}_{r \in R} \). We show here that there exists a steady schedule \( \tau \) such that \( \tau(\sigma) = \sigma' \), using the following idea: (i) we construct a configuration \( \sigma'' \) that is reachable from \( \sigma \) in one step, i.e., we show that there is a transition \( t \) such that \( t(\sigma) = \sigma'' \), and (ii) we prove that \( \phi_{steady}(\sigma'', \sigma') \) holds for an assignment \( \{z_r\}_{r \in R} \) such that \( \{z_r\}_{r \in R} < \{y_r\}_{r \in R} \). By applying our induction hypothesis, there is a schedule \( \rho \) with \( \rho(\sigma'') = \sigma' \). We would then have \( \tau = t \cdot \rho \) as the required schedule.

Let us construct \( \sigma'' \). By the assumption, there is a rule \( r' \in R \) such that \( y_{r'} > 0 \). As \( \phi_{steady}(\sigma, \sigma') \) holds with the assignment \( Y = \{y_r\}_{r \in R} \), \( \phi_{app}(\sigma, Y) \) is true and therefore there must be a set of rules \( S = \{r_1, \ldots, r_s\} \subseteq R \) such that \( \phi'_{chain}(S, \sigma, Y) \) holds, that is,

\[
\sigma.S[r_1\text{from}] > 0 \land \bigwedge_{1 \leq i \leq s} y_{r_i} > 0 \land \bigwedge_{1 < i \leq s} r_{i-1}.to = r_i\text{from} \land r_s = r'
\]

Let \( r = r_1 \) and let \( \ell = r\text{from} \). If there is no fireable cycle at \( \ell \) with respect to \( Y \), then by Lemma 2 we have that \( \phi_{steady}(r(\sigma), \sigma') \) is true with an assignment.
We consider three cases: there exists a set of rules \( \phi \) and \( \sigma \) such that \( \phi \) is required to hold and we have \( \phi_R(\sigma, Y) \) is true with an assignment \( Z < Y \) and so we can define \( \sigma'' = t_1(\sigma) \). Hence we can construct the required \( \sigma'' \) in either case and so the proof is complete.

Now we will prove the Lemmas \( \mathcal{B} \) and \( \mathcal{X} \).

**B.2 Proof of Lemma 2**

Proof. Let \( t := t.\text{from} \). Since \( y_r > 0 \), from \( \phi_R(\sigma, Y) \) we obtain that \( \sigma \models t.\varphi \). As we have \( \sigma.\mathcal{K}[t.\text{from}] = \sigma.\mathcal{K}[t] > 0 \) we know that \( t \) is enabled at \( \sigma \). Hence we set \( \sigma'' = t(\sigma) \) and let \( Z = \{ z_r \}_{r \in \mathcal{R}} \) be the following assignment: \( z_r = y_r \) if \( r \neq t \) and \( z_r = y_r - 1 \) if \( r = t \). It can be easily checked that all of the following holds: \( \phi_{\text{base}}(\sigma'', \sigma') \land \phi_{\mathcal{L}}(\sigma'', \sigma', Z) \land \phi_T(\sigma'', \sigma', \mathcal{Z}) \land \phi_R(\sigma'', \mathcal{Z}) \). Therefore, we only need to prove that \( \phi_{\text{appl}}(\sigma'', Z) \) holds.

Fix any arbitrary rule \( r \in \mathcal{R} \) such that \( z_r > 0 \). We now have to show that there exists a set of rules \( S' = \{ r'_1, \ldots, r'_s \} \subseteq \mathcal{R} \) such that

\[
\sigma''.\mathcal{K}[r'_1.\text{from}] > 0 \land \bigwedge_{1 \leq i \leq s'} z_{r'_i} > 0 \land \bigwedge_{1 < i \leq s'} r'_{i-1}.\text{to} = r'_{i}.\text{from} \land r'_{s} = r
\]

Since \( z_r > 0 \), it follows that \( y_r > 0 \). Since \( \phi_{\text{appl}}(\sigma, Y) \) holds, it follows that there exists a set of rules \( S = \{ r_1, \ldots, r_s \} \subseteq \mathcal{R} \) such that

\[
\sigma.\mathcal{K}[r_1.\text{from}] > 0 \land \bigwedge_{1 \leq i \leq s} y_{r_i} > 0 \land \bigwedge_{1 < i \leq s} r_{i-1}.\text{to} = r_i.\text{from} \land r_s = r
\]

We consider three cases:

- Suppose \( \sigma''.\mathcal{K}[r'_1.\text{from}] > 0 \) and suppose \( z_{r_i} > 0 \) for every \( 1 \leq i \leq s \). Then we can simply set \( S' := S \).
- Suppose there exists \( i \) such that \( z_{r_i} = 0 \). Hence \( z_{r_i} \neq y_{r_i} \) and this immediately implies that \( r_i = t \). Since \( \sigma'' \) was obtained from \( \sigma \) by firing \( t \), it follows that \( \sigma''.\mathcal{K}[t.\text{to}] = \sigma''.\mathcal{K}[r_i.\text{to}] = \sigma''.\mathcal{K}[r_{i+1}.\text{from}] > 0 \). It is then easy to see that we can set \( S' := \{ r_{i+1}, \ldots, r_s \} \).
- Suppose \( \sigma''.\mathcal{K}[r_1.\text{from}] = 0 \). We can assume that \( z_{r_i} > 0 \) for every \( 1 \leq i \leq s \) as otherwise the previous case applies. Since \( \sigma'' \) was obtained from \( \sigma \) by firing \( t \), it follows that \( r_1.\text{from} = t.\text{from} = \ell \). Let \( \text{in}^t_1, \ldots, \text{in}^t_e \) and \( \text{out}^t_1, \ldots, \text{out}^t_f \) be the set of incoming and outgoing rules at \( \ell \) in the threshold automaton.

We further split this into two more cases:

- There exists \( i \) such that \( z_{\text{in}_i^t} > 0 \). Let \( r'' = \text{in}^t_i \). Since \( z_{r''} > 0 \) it follows that \( y_{r''} > 0 \). Hence there exists a set \( S'' = \{ r''_1, \ldots, r''_s \} \subseteq \mathcal{R} \) such that

\[
\sigma.\mathcal{K}[r''_1.\text{from}] > 0 \land \bigwedge_{1 \leq i \leq s''} y_{r''_i} > 0 \land \bigwedge_{1 < i \leq s''} r''_{i-1}.\text{to} = r''_i.\text{from} \land r''_{s''} = r''
\]

Since \( \ell \) is not part of any fireable cycle with respect to \( Y \) and since \( r''.\text{to} = \ell \) it follows that \( r''_1.\text{from} \neq \ell \). Hence \( \sigma.\mathcal{K}[r''_1.\text{from}] > 0 \implies \)
We now consider three cases: there exists a set of rules \( \phi \). Therefore, we only need to prove that \( \sigma \), \( a \subseteq R \).

- There does not exist \( i \) such that \( z_{in} > 0 \). We show that this case cannot happen. Recall that the assignment \( Z \) satisfies the formula \( \phi_L(\sigma'', \sigma', Z) \). In particular,

\[
\sum_{i=1}^{a} z_{in} - \sum_{i=1}^{b} z_{out} = \sigma' \cdot \kappa[\ell] - \sigma'' \cdot \kappa[\ell]
\]

By assumption, \( z_{in} = 0 \) for all \( i, 1 \leq i \leq a \). Also \( z_{a} > 0 \) and \( r_{a} \) is (by definition) an outgoing transition from \( l \). Hence the left hand side of the equation is strictly less than \( 0 \). Since \( \sigma'' \cdot \kappa[\ell] = 0 \), it follows that the right hand side of the equation is bigger than or equal to \( 0 \), which gives rise to a contradiction.

**B.3 Proof of Lemma 3**

*Proof.* Suppose \( \ell \) is part of some fireable cycle with respect to \( Y \), i.e., there exists rules \( T = \{t_{1}, \ldots, t_{m}\} \) such that \( y_{t_{1}} > 0, \ldots, y_{t_{m}} > 0, t_{i}.to = t_{i+1}.from \) for all \( i < m \) and \( t_{m}.to = t_{1}.from = \ell \). Let \( l = t_{1} \). Since \( y_{t} > 0 \), from \( \phi_{R}(\sigma, Y) \) we obtain that \( \sigma \models L \). As we have \( \sigma \cdot \kappa[\ell.first] = \sigma \cdot \kappa[\ell] > 0 \) we know that \( l \) is enabled at \( \sigma \). Hence we set \( \sigma'' = t(\sigma) \) and let \( Z = \{z_{r} \}_{r \in R} \) be the following assignment: \( z_{r} = y_{r} \) if \( r \neq t \) and \( z_{r} = y_{r} - 1 \) if \( r = t \). It can be easily checked that all of the following holds: \( \phi_{base}(\sigma'', \sigma') \land \phi_{L}(\sigma'', \sigma', Z) \land \phi_{of}(\sigma'', \sigma', Z) \land \phi_{R}(\sigma'', Z) \). Therefore, we only need to prove that \( \phi_{appl}(\sigma'', Z) \) holds.

Fix any arbitrary rule \( r \in R \) such that \( z_{r} > 0 \). We now have to show that there exists a set of rules \( S' = \{r_{1}', \ldots, r_{s}'\} \subseteq R \) such that

\[
\sigma'' \cdot \kappa[r'_{1}.from] > 0 \land \bigwedge_{1 < i \leq s'} r'_{i-1}.to = r'_{i}.from \land r'_{s} = r
\]

Since \( z_{r} > 0 \), it follows that \( y_{r} > 0 \). Since \( \phi_{appl}(\sigma, Y) \) holds, it follows that there exists a set of rules \( S = \{r_{1}, \ldots, r_{s}\} \subseteq R \) such that

\[
\sigma \cdot \kappa[r_{1}.from] > 0 \land \bigwedge_{1 < i \leq s} r_{i-1}.to = r_{i}.from \land r_{s} = r
\]

We now consider three cases:

- Suppose \( \sigma'' \cdot \kappa[r_{1}.from] > 0 \) and suppose \( z_{r_{i}} > 0 \) for every \( 1 \leq i \leq s \). Then we can simply set \( S' := S \).
- Suppose there exists \( i \) such that \( z_{r_{i}} = 0 \). Hence \( z_{r_{i}} \neq y_{r_{i}} \), and this immediately implies that \( r_{i} = t \). Since \( \sigma'' \) was obtained from \( \sigma \) by firing \( t \), it follows that \( \sigma'' \cdot \kappa[t.to] = \sigma'' \cdot \kappa[r_{i}.to] = \sigma'' \cdot \kappa[r_{i+1}.from] > 0 \). It is then easy to see that we can set \( S' := \{r_{i+1}, \ldots, r_{s}\} \).
– Suppose \( \sigma'' \cdot \kappa[r_\ell, \text{from}] = 0 \). We can assume that \( z_{r_i} > 0 \) for every \( 1 \leq i \leq s \) as otherwise the previous case applies. Since \( \sigma'' \) was obtained from \( \sigma \) by firing \( t \), it follows that \( r_1.\text{from} = t.\text{from} = \ell = t_m.\text{to} \). Notice that \( \sigma'' \cdot \kappa[t.\text{to}] = \sigma'' \cdot \kappa[t_2.\text{from}] \geq 0 \) and also that \( z_{t_i} = y_{t_i} > 0 \) for all \( i > 1 \). Hence we can set \( S' := \{t_2, \ldots, t_m, r_1, \ldots, r_s\} \).

**Remark:** Notice that, in the course of proving Theorem 4, we have actually proved something much stronger. For a schedule \( \tau \) it is easy to notice the following:

**Lemma 4.** If \( \tau \) is a steady schedule between \( \sigma \) and \( \sigma' \) then \( \phi_{\text{steady}}(\sigma, \sigma') \) holds with assignment \( \{\tau(r)\}_{r \in \mathcal{R}} \). Further, if \( \phi_{\text{steady}}(\sigma, \sigma') \) holds with assignment \( \{y_r\}_{r \in \mathcal{R}} \) then it is possible to create a steady schedule \( \tau \) between \( \sigma \) and \( \sigma' \) such that \( \tau(r) = y_r \) for all rules \( r \).

**B.4 Proof of Theorem 5**

**Proof.** Let \( \sigma \) and \( \sigma' \) be two configurations. First we will show that deciding if there is a steady schedule \( \tau \) such that \( \tau(\sigma) = \sigma' \) can be done in NP. From Theorem 4 we know there is a steady schedule from \( \sigma \) to \( \sigma' \) if and only if the existential Presburger formula \( \phi_{\text{steady}}(\sigma, \sigma') \) is true. However the formula \( \phi_{\text{steady}}(\sigma, \sigma') \) is exponential in size when constructed naively. But the exponential dependence comes only from the constraint \( \phi_{\text{appl}} \) which can be easily reduced to a polynomial dependence as follows: For each rule \( t \) we guess a set \( S = \{r^t_1, \ldots, r^t_s\} \) and check that \( r^t_i = t \) and \( r^t_i.\text{from} = r^t_{i+1}.\text{to} \) for all \( i < s \). Once that is done, we replace \( \phi_{\text{appl}} \) with

\[
\bigwedge_{t \in \mathcal{R}} x_t > 0 \implies \sigma[r^t_i.\text{from}] > 0 \land x_{r^t_i} > 0 \land \cdots \land x_{r^t_s} > 0
\]

It is then clear that \( \phi_{\text{steady}}(\sigma, \sigma') \) is true if and only if at least one of our guesses is true. Now we have reduced it to a polynomial sized existential Presburger arithmetic formula which we can decide in NP (See [14]). This shows that checking whether there is a steady schedule from \( \sigma \) to \( \sigma' \) is in NP.

Now suppose we want to check if there is a (general) run from \( \sigma \) to \( \sigma' \). By Proposition 4 every path can be written as the concatenation of at most \( K \) steady paths where \( K = |\phi_{\text{rise}}| + |\phi_{\text{fall}}| + 1 \). Hence it suffices to check if there is a run of the form

\[
\sigma = \sigma_0 \rightarrow^* \sigma'_0 \rightarrow \sigma_1 \rightarrow^* \sigma'_1 \rightarrow \sigma_2 \ldots \sigma_K \rightarrow^* \sigma'_K = \sigma'
\]

such that the context of each \( \sigma_i \) is the same as the context of \( \sigma'_i \). With this in mind, we define \( \phi_{\text{step}}(\eta, \eta') \) as the formula \( \phi_{\text{steady}}(\eta, \eta') \) except we do not enforce that \( \eta \) and \( \eta' \) have the same context and we enforce that the existential variables \( \{x_r\}_{r \in \mathcal{R}} \) obey the constraint \( \sum_{r \in \mathcal{R}} x_r \leq 1 \). It is easy to see that \( \phi_{\text{step}}(\eta, \eta') \)
is true iff \( \eta' \) can be reached from \( \eta \) in at most one step. Hence if we define \( \phi_{\text{reach}}(\sigma, \sigma') \) to be

\[
\exists \sigma_0, \sigma'_0, \ldots, \sigma_K, \sigma'_K \left( \sigma_0 = \sigma \land \sigma'_K = \sigma' \land \bigwedge_{0 \leq i \leq K} \phi_{\text{steady}}(\sigma_i, \sigma'_i) \land \bigwedge_{0 \leq i \leq K-1} \phi_{\text{step}}(\sigma'_i, \sigma_{i+1}) \right)
\]

then it is clear that there is a run from \( \sigma \) to \( \sigma' \) iff \( \phi_{\text{reach}}(\sigma, \sigma') \) is satisfied. To decide if \( \phi_{\text{reach}}(\sigma, \sigma') \) is true, we eliminate the exponentially sized disjunctions in \( \phi_{\text{steady}} \) as before and check that the resulting (polynomial sized) formula is satisfiable.

The proof of Theorem 5 is complete. However, for future purposes, we modify the formula \( \phi_{\text{reach}}(\sigma, \sigma') \) slightly, so that it holds some extra information. Notice that in equation 3, for each rule \( r \) corresponding to the first configuration is fixed to \( \sigma \), the formula \( \phi_{\text{steady}}(\sigma_i, \sigma'_i) \) and there is also an existential variable \( y'_i \) appearing in the subformula \( \phi_{\text{step}}(\sigma'_i, \sigma_{i+1}) \). We introduce a new existential variable \( \sum_r \) and enforce the constraint \( \sum_r = \sum_{i=0}^K x'_i + \sum_{i=0}^{K-1} y'_i \). Using Lemma 4 we can easily prove that

**Lemma 5.** The following are true:

- Suppose \( \phi_{\text{reach}}(\sigma, \sigma') \) is true with the assignment \( \{ z_r \}_{r \in \mathcal{R}} \) to the existential variables \( \{ \sum_r \}_{r \in \mathcal{R}} \). Then there is a schedule \( \tau \) between \( \sigma \) and \( \sigma' \) such that \( \tau(r) = z_r \) for all rules \( r \).
- Conversely, if \( \tau \) is a schedule between \( \sigma \) and \( \sigma' \), then it is possible to satisfy \( \phi_{\text{reach}}(\sigma, \sigma') \) by setting \( \{ \tau(r) \}_{r \in \mathcal{R}} \) to the variables in \( \{ \sum_r \}_{r \in \mathcal{R}} \).

For liveness properties, this lemma will prove very useful.

### B.5 Proof of Corollary 1

**Proof.** Recall that, in the parameterized reachability problem, we are given two sets \( \mathcal{L}_{=0}, \mathcal{L}_{>0} \) of locations, and we want to decide if there is an initial configuration \( \sigma_0 \), and some configuration \( \sigma \) reachable from \( \sigma_0 \) such that it satisfies \( \sigma.\kappa[\ell] = 0 \) for every \( \ell \in \mathcal{L}_{=0} \) and \( \sigma.\kappa[\ell] > 0 \) for every \( \ell \in \mathcal{L}_{>0} \). The non-parameterized reachability problem is similar, except the initial configuration \( \sigma_0 \) is also given as part of the input.

By Theorem 5, we know that there is a formula \( \phi_{\text{reach}} \) in existential Presburger arithmetic with \( (2|\mathcal{L}| + 2|\Gamma| + 2|\Pi|) \) free variables such that \( \phi_{\text{reach}}(\sigma, \sigma') \) is true iff there is a run between \( \sigma \) and \( \sigma' \). For parameterized reachability, we modify the formula \( \phi_{\text{reach}} \) so that the free variables become existential variables. For non-parameterized reachability, we modify \( \phi_{\text{reach}} \) so that the free variables corresponding to the first configuration is fixed to \( \sigma_0 \) and the free variables corresponding to the second configuration become existential variables. Finally, in both the cases, we specify that the second configuration must satisfy the constraints according to the sets \( \mathcal{L}_{=0} \) and \( \mathcal{L}_{>0} \). It is then clear that by checking if the constructed formula is satisfiable, we can solve both the problems in NP.
We break down the proof of Theorem 6 into various parts. As a preliminary step, we introduce the notion of a lasso path.

### C.1 Lasso paths and cut graphs

Similar to model checking finite systems, we first show that if a path of the threshold automaton satisfies the \( \text{ELTL}_{\mathcal{FT}} \) formula \( \varphi \), then there is an “ultimately periodic” path which also satisfies \( \varphi \).

For a schedule \( \tau \) let \([\tau]\) denote the set of all rules which appear in \( \tau \). We have the following definition:

**Definition 6.** A path \((\sigma, \tau')\) is called a lasso path if \( \tau' \) can be decomposed as \( \rho \circ \tau \omega \) such that if we let \( \sigma_1 = \rho(\sigma) \) and \( \sigma_2 = (\rho \circ \tau)(\sigma) \) then

- \( \sigma_1 \cdot \kappa = \sigma_2 \cdot \kappa \).
- If \( r \in [\tau] \) such that \( r \cdot u[x] > 0 \) then for all rules \( r' \in [\tau] \), \( x \) does not appear in any of the fall guards of \( r' \).

**Proposition 2.** If \((\sigma', \tau')\) satisfies a formula \( \varphi \) in \( \text{ELTL}_{\mathcal{FT}} \) then there is a lasso path which satisfies \( \varphi \).

**Proof.** We do not give details on Buchi automata and product construction, as these are well-known.

The formula \( \varphi \) in \( \text{ELTL}_{\mathcal{FT}} \) can be thought of as an LTL formula over the atomic propositions \( p_f \) and hence there is a Buchi automaton \( B_{\varphi} = (\Sigma, 2^\Sigma, Q_0, \Delta, F) \) which recognizes exactly those sequence of propositions which satisfy \( \varphi \).

Fix a threshold automaton \( \mathcal{T}A = (\mathcal{L}, \mathcal{I}, \mathcal{V}, \mathcal{R}) \). Let \( \Sigma \) be the set of all configurations of \( \mathcal{T}A \), \( I \) be the set of all initial configurations of \( \mathcal{T}A \) and \( T \subseteq \Sigma \times \mathcal{R} \times \Sigma \) be the transition relation between the configurations. Let \( \text{Sys}(\mathcal{T}A) = (\Sigma, I, T) \).

We can then construct the standard product construction \( \text{Sys}(\mathcal{T}A) \times B_{\varphi} \), where there is a transition \(((\sigma, q), p, (\sigma', q'))\) iff \((q, p, q') \in \Delta, (\sigma, r, \sigma') \in T \) for some rule \( r \) and \( \sigma \models p \). A path \((\sigma_0, q_0), (\sigma_1, q_1), \ldots \) is an accepting path iff it visits a state in \( \Sigma \times F \) infinitely often.

Let \( PA = (\sigma_0, q_0), (\sigma_1, q_1), \ldots \) be an infinite accepting path in \( \text{Sys}(\mathcal{T}A) \times B_{\varphi} \). This gives rise to a path \( P = \sigma_0, r_0, \sigma_1, r_1, \ldots \) in \( \text{Sys}(\mathcal{T}A) \). Let \( V \subseteq \Gamma \) be the set of all variables which are infinitely often incremented along \( P \). Since these variables are infinitely often incremented, it follows that there exists \( i \) such that for all \( j \geq i \), \( r_j \) does not have any fall guards with any of the variables in \( V \). Since the number of processes does not change during the course of a path and since the number of states in \( F \) is finite, by combining all of the above facts, it follows that we can find an infinite subsequence of \( PA \) of the form \((\sigma_{i_1}, q_{i_1}), (\sigma_{i_2}, q_{i_2}), \ldots \) such that

- \( q = q_{i_1} = q_{i_2} = \ldots \) and \( q \in F \).
\[ \sigma_i \land \phi_i = \sigma_{i+1} \land \phi_{i+1} = \ldots \]
- For every \( i \), \( \sigma_i \land \phi_i < \sigma_{i+1} \land \phi_{i+1} \) iff \( x \in V \)
- For every \( i_1 \leq j \leq i_2 \), the rule \( r_j \) does not have any fall guards with any of the variables in \( V \).

Let \( \rho = r_1 \circ r_2 \circ \cdots \circ r_{i-1} \) and let \( \tau = r_{i} \circ r_{i+1} \circ \cdots \circ r_{i-1} \). It can be verified (using all the points given above) that \( \text{path}(\sigma_0, \rho \circ \tau) \) is a lasso path which satisfies \( \phi \).

Hence it suffices to concentrate on lasso paths in the future. To prove our theorem, we also need the notion of a cut graph of a formula \( \phi \). However, for our purposes, it suffices to know that the cut graph \( Gr(\phi) \) of a formula \( \phi \) is a directed acyclic graph which consists of two distinguished vertices \( \phi \) and \( \Phi \).

For a formula \( \phi \equiv \phi_0 \land F \phi_1 \land \cdots \land F \phi_k \land G \phi_{k+1} \) in normal form, let \( \text{prop}(\phi) = \phi_0 \). The following lemma (proved in [19]) connects the notion of a lasso path and the cut graph. Intuitively, it says that a lasso path satisfies a formula \( \phi \) iff it can be “cut” into finitely many pieces such that (1) the endpoints of each piece satisfy some (propositional) formula dictated by the cut-graph and (2) all the configurations between any two endpoints satisfy some other (propositional) formula as dictated by the cut-graph.

**Lemma 6.** A lasso path \( \text{path}(\sigma_0, \rho \circ \tau) \) satisfies a formula \( \phi \equiv \phi_0 \land F \phi_1 \land \cdots \land F \phi_k \land G \phi_{k+1} \) iff there is a topological ordering
\[
v_1, v_2, \ldots, v_c, v_{c-1}, v_c = \text{loop}_{\text{start}}, v_{c+1}, \ldots, v_{l-1}, v_l = \text{loop}_{\text{end}}\]
of its cut graph and a finite path
\[
\sigma_0, \tau_0, \sigma_1, \tau_1, \ldots, \sigma_c, \tau_c, \ldots, \sigma_l-1, \tau_{l-1}, \sigma_l
\]
such that the following holds: If each \( v_i \) (other than \( v_c \) and \( v_l \)) is of the form \( \phi_0 \land F \phi_1 \land \cdots \land F \phi_k \land G \phi_{k+1}, \) then
- \( \sigma_0 \) is an initial configuration, and \( \tau_i(\sigma_i) = \sigma_{i+1} \) for every \( 0 \leq i < l \).
- \( \tau_0 \circ \tau_1 \circ \cdots \circ \tau_{c-1} = \rho \circ \tau^K \) for some \( K \) and \( \tau_c \circ \cdots \circ \tau_{l-1} = \tau \).
- \( \sigma_0 \vdash \phi_0 \) and \( \text{Cfgs}(\sigma_0, \tau_0) \vdash \text{prop}(\phi_{k+1}) \).
- For every \( i \not= \{ c, l \} \), we have \( \sigma_i \vdash \phi_i \).
- If \( i < c \), then \( \text{Cfgs}(\sigma_i, \tau_i) \vdash \bigwedge_{0 \leq j \leq l} \text{prop}(\phi_{k+1}) \).
- If \( i \geq c \), then \( \text{Cfgs}(\sigma_i, \tau_i) \vdash \bigwedge_{0 \leq j < l} \text{prop}(\phi_{k+1}) \).

We note the subtle but important difference in the indices between the last two points. When \( i < c \), we require \( \text{Cfgs}(\sigma_i, \tau_i) \vdash \bigwedge_{0 \leq j \leq l} \text{prop}(\phi_{k+1}) \), but when \( i \geq c \), we require \( \text{Cfgs}(\sigma_i, \tau_i) \vdash \bigwedge_{0 \leq j < l} \text{prop}(\phi_{k+1}) \).

**Remark:** In [19] the above lemma was only proven for automata where there are no updates in a cycle, but virtually the same proof also holds for the general case.

Combining Proposition 2 and Lemma 6, we get the following lemma.

\[ 28 \]
Corollary 2. The formula \( \varphi \equiv \phi_0 \land F \phi_1 \land \cdots \land F \phi_k \land G \phi_{k+1} \) is satisfiable if and only if there is a topological ordering \( <_{Gr} \)

\[
v_1, v_2, \ldots, v_{c-1}, v_c = \text{loop}_{st}, v_{c+1}, \ldots, v_{l-1}, v_l = \text{loop}_{end}
\]

of its cut graph and a finite path \( P \)

\[
\sigma_0, \tau_0, \sigma_1, \tau_1, \ldots, \sigma_l, \tau_l, \sigma_{l-1}, \tau_{l-1}, \sigma_l
\]

such that the following holds: If each \( v_i \) (other than \( v_c \) and \( v_l \)) is of the form

\[
\phi_0^i \land F \phi_1^i \land \cdots \land F \phi_k^i \land G \phi_{k+1}^i,
\]

- \( \sigma_0 \) is an initial configuration, \( \sigma_c, \kappa = \sigma_l, \kappa \) and \( \tau_i(\sigma_i) = \sigma_{i+1} \) for every \( i < l \).
- If \( r \in [\tau_c \circ \tau_{c+1} \cdots \circ \tau_{l-1}] \) such that \( r.\mathbf{u}[x] > 0 \) then for every rule \( r' \in [\tau_c \circ \tau_{c+1} \cdots \circ \tau_{l-1}] \), \( x \) does not appear in any of the fall guards of \( r' \).
- \( \sigma_0 \models \phi_0 \) and \( \text{Cfgs}(\sigma_0, \tau_0) \models \text{prop}(\phi_{k+1}) \).
- For every \( i \notin \{c, l\} \), we have \( \sigma_i \models \phi_0^i \).
- If \( i < c \), then \( \text{Cfgs}(\sigma_i, \tau_i) \models \bigwedge_{0 \leq j < i} \text{prop}(\phi_{k+1}^j) \).
- If \( i \geq c \), then \( \text{Cfgs}(\sigma_i, \tau_i) \models \bigwedge_{0 \leq j < i} \text{prop}(\phi_{k+1}^j) \).

If such a topological ordering \( <_{Gr} \) and such a path \( P \) exists then we call the pair \( (<_{Gr}, P) \) a witness to the formula \( \varphi \).

This finishes the first part of the proof of theorem.6

C.2 Roadmap

Before we move on to the next part, we give some intuition. Notice that for \( (<_{Gr}, P) \) to be a witness for \( \varphi \), among other restrictions, it has to satisfy some reachability conditions \( (\sigma_{i+1} = \tau_i(\sigma_i)) \), some safety conditions \( (\sigma_i \models \phi) \) as well as some liveness conditions \( (\text{Cfgs}(\sigma_i, \tau_i) \models \phi) \). Our strategy for the next part is the following: Intuitively, the main hindrance for us to apply our main result (that the reachability relation is existential Presburger definable) here are the liveness conditions. Here is where the multiplicativity condition comes into picture. For a propositional formula \( p \), we will define an existential Presburger arithmetic formula \( \phi_p \) with the following properties:

- If there is a path from \( \sigma \) to \( \sigma' \) such that all the configurations in the path satisfy \( p \) then \( \phi_p(\sigma, \sigma') \) will be true.
- If \( \phi_p(\sigma, \sigma') \) is true then there is a path from \( 2 \cdot \sigma \) to \( 2 \cdot \sigma' \) such that all the configurations in the path satisfy \( p \).

Using this newly defined formula \( \phi_p \) and some additional tricks, we then show that for a topological ordering \( <_{Gr} \), we can write a formula \( \phi_{\text{live}} \) such that \( \phi_{\text{live}} \) will be satisfied if there is a path \( P \) such that \( (<_{Gr}, P) \) is a witness to the given specification.
C.3 The usefulness of multiplicativity

First, given a path $P$ from $\sigma_0$ to $\sigma_m$, we define a lifted path $\text{lift}(P)$ between $2 \cdot \sigma$ and $2 \cdot \sigma_m$.

**Definition 7.** Suppose $\sigma_0, \tau_0, \sigma'_0, t_0, \sigma_1, \tau_1, \sigma'_1, t_1, \ldots, \sigma_{m-1}, \tau_{m-1}, \sigma'_{m-1}, t_{m-1}, \sigma_m$ is a path such that each $\tau_i$ is a steady schedule between $\sigma_i$ and $\sigma_{i+1}$ and each $t_i$ is a rule such that $\sigma'_i$ and $\sigma_{i+1}$ have different contexts. Its corresponding lifted path $\text{lift}(P)$ is defined by $2 \cdot \sigma_0, 2 \cdot \tau_0, 2 \cdot \sigma'_0, 2 \cdot t_0, 2 \cdot \sigma_1, 2 \cdot \tau_1, \ldots, 2 \cdot t_{m-1}, 2 \cdot \sigma_m$.

Using multiplicativity it is easy to verify that $\text{lift}(P)$ is a valid path from $2 \cdot \sigma_0$ to $2 \cdot \sigma_m$.

We say that a path $P = \sigma_0, \tau_0, \sigma_1, \ldots, \sigma_m, \tau_m$ satisfies a proposition $p$ if all the configurations in $P$ satisfy $p$. We now prove the following lemma:

**Lemma 7.** Let $p$ be a propositional pf-formula as given in ELTL$_{\text{FT}}$. Then there exists an existential Presburger formula $\phi_p$ such that

- If there is a path $P$ from $\sigma$ to $\sigma'$ such that $P$ satisfies $p$ then $\phi_p(\sigma, \sigma')$ will be true.
- If $\phi_p(\sigma, \sigma')$ is true then there is a path $P$ from $\sigma$ to $\sigma'$ such that the lifted path $\text{lift}(P)$ from $2 \cdot \sigma$ to $2 \cdot \sigma'$ satisfies $p$.

**Proof.** Recall that by theorem 5 that the reachability relation $\phi_{\text{reach}}(\sigma, \sigma')$ is

$$\exists \sigma_0, \sigma'_0, \ldots, \sigma_K, \sigma'_K \left( \sigma_0 = \sigma \land \sigma'_K = \sigma' \land \bigwedge_{0 \leq i \leq K} \phi_{\text{steady}}(\sigma_i, \sigma'_i) \land \bigwedge_{0 \leq i \leq K-1} \phi_{\text{step}}(\sigma'_i, \sigma_{i+1}) \right)$$

where $K = |\Phi| + 1$. $\phi_{\text{steady}}(\sigma_i, \sigma'_i)$ is true if there is a steady schedule between $\sigma_i$ and $\sigma'_i$ and $\phi_{\text{step}}(\sigma'_i, \sigma_{i+1})$ is true iff $\sigma_{i+1}$ can be reached from $\sigma'_i$ in at most one step. Further recall that for each rule $r$ there is an existential variable $\text{sum}_r$, which denotes the number of times the rule $r$ is fired in the run between $\sigma$ and $\sigma'$.

Let $p$ be a pf-formula as given in ELTL$_{\text{FT}}$. Depending on the structure of $p$ we introduce a different formula for $\phi_p(\sigma, \sigma')$: In each case, it will be immediately clear that if there is a path $P$ from $\sigma$ to $\sigma'$ satisfying $p$, then $\phi_p(\sigma, \sigma')$ is true. For this reason, we will only discuss how to prove the other direction of the claim (namely, if $\phi_p(\sigma, \sigma')$ is true then there is a path $P$ from $\sigma$ to $\sigma'$ such that the lifted path $\text{lift}(P)$ from $2 \cdot \sigma$ to $2 \cdot \sigma'$ satisfies $p$.)

- $p := (S = 0)$ for some $S \subseteq \mathcal{L}$. In this case, we construct $\phi_p$ from $\phi_{\text{reach}}$ as follows: To each of the formulas $\phi_{\text{steady}}(\sigma_i, \sigma'_i)$, we add the following constraint

$$\bigwedge_{\ell \in S} \sigma_i, K[\ell] = 0 \land \bigwedge_{\ell \in S} \sigma'_i, K[\ell] = 0 \land \bigwedge_{\ell \in S} \sum_r \text{count}_{\ell} = 0 \quad (4)$$

where $In_{\ell}$ denotes the set of all incoming transitions to $\ell$. By using Lemma 5 it is clear that if $\phi_p(\sigma, \sigma')$ is true then there is a path $P$ between $\sigma$ and $\sigma'$.
such that all the configurations in the path $P$ satisfy $p$. Consequently, it is then easy to verify that all the configurations in the lifted path $\text{lift}(P)$ also satisfy $p$.

- $p := \neg(S = 0)$ for some $S \subseteq L$. In this case, we construct $\phi_p$ from $\phi_{\text{reach}}$ as follows: To each of the formulas $\phi_{\text{steady}}(\sigma_i, \sigma'_i)$ we add the following constraint

$$\bigvee_{\ell \in S} \sigma_i[\ell] > 0 \land \bigvee_{\ell \in S} \sigma'_i[\ell] > 0$$

(5)

Using Lemma 5 it is then clear that if $\phi_p(\sigma, \sigma')$ is true then there exists a path between $\sigma$ and $\sigma'$ such that each of the intermediate configurations $\sigma_0, \sigma'_0, \ldots, \sigma_K, \sigma'_K$ satisfy $p$. By a careful inspection of the corresponding lifted path $\text{lift}(P)$ it can be verified that every configuration in the lifted path satisfies $p$.

- $p := (S' = 0) \land \bigwedge_{S \in T} \neg(S = 0)$. In this case, to each $\phi_{\text{steady}}(\sigma_i, \sigma'_i)$ we add

$$\bigwedge_{\ell \in S'} \sigma_i[\ell] = 0 \land \bigwedge_{\ell \in S'} \sum_r = 0 \land$$

$$\left( \bigwedge_{S \in T} \bigvee_{\ell \in S} \sigma_i[\ell] > 0 \right) \land \left( \bigwedge_{S \in T} \bigvee_{\ell \in S} \sigma'_i[\ell] > 0 \right)$$

(6)

- $p := (g \Rightarrow c)$ where $g$ is a $gf$-formula and $c := (S' = 0) \land \bigwedge_{S \in T} \neg(S = 0)$ is a $cf$-formula. In this case, to each $\phi_{\text{steady}}(\sigma_i, \sigma'_i)$ we add the constraint

$$(\sigma_i \models g) \Rightarrow \phi_c$$

(7)

where $\phi_c$ is the constraint given by equation (6).

Given a path $P$, let $P(r)$ denote the number of times the rule $r$ appears in $P$. Notice that once again we have proved something stronger.

**Lemma 8.** Let $p$ be a propositional formula in $ELTL_{\text{FT}}$. Then,

- If there is a path $P$ between $\sigma$ and $\sigma'$ satisfying $p$ then it is possible to satisfy $\phi_p(\sigma, \sigma')$ by setting $\{P(r)\}_{r \in R}$ to the variables $\{\sum_r\}_{r \in R}$.
- Suppose $\phi_p(\sigma, \sigma')$ is satisfiable. Then there is path $P$ between $\sigma$ and $\sigma'$ such that $P(r) = \sum_r$ for every rule $r$ and the lifted path $\text{lift}(P)$ satisfies $p$.

**C.4 Proof of Theorem 6**

Using the above lemma, we now give a proof of Theorem 6. The proof proceeds by encoding the conditions mentioned in Corollary 2.

Let $\varphi \equiv \phi_0^0 \land \mathbf{F} \phi_0^0 \land \cdots \land \mathbf{F} \phi_{k_0}^0 \land \mathbf{G} \phi_{k_0+1}^0$. We want to guess a witness $(<_{Gr}, P)$. For this reason, we first guess and fix a topological ordering $<_{Gr}$

$$v_1, v_2, \ldots, v_c = \text{loopstart}, v_{c+1}, \ldots, v_l = \text{loopend}$$

31
of the cut graph \( Gr(\varphi) \). Let each \( v_i \) (apart from \( v_c \) and \( v_l \)) be of the form 
\[ \phi_0^i \land F \phi_1^i \land \cdots \land F \phi_k^i \land G \phi_{k+1}^i. \]

We will write a formula \( \phi_{live} \) with existential variables describing \( l+1 \) configurations \( \sigma_0, \sigma_1, \ldots, \sigma_c, \sigma_i, \ldots, \sigma_l \). We will now show to enforce each of the conditions mentioned by Corollary 2 as a formula in existential Presburger arithmetic. (For ease of presentation, we provide the steps in an order different from the one in Corollary 2)

**First condition:** \( \sigma_0 \) is an initial configuration and \( \sigma_c, \kappa = \sigma_i, \kappa \): Clearly this can be encoded as an existential Presburger formula.

**Second condition:** For each \( i \notin \{c, l\} \), \( \sigma_i \models \phi_0^i \): Since each \( \phi_0^i \) is a propositional \( pf \) formula, this constraint can once again be encoded as an existential Presburger formula.

**Third condition:** For each \( i < c \), the path between \( \sigma_i \) and \( \sigma_{i+1} \) must satisfy
\[ \bigwedge_{0 \leq j \leq i} \text{prop}(\phi_{k+1}^j) \]: Let \( p_i := \bigwedge_{0 \leq j \leq i} \text{prop}(\phi_{k+1}^j) \). We enforce this condition by putting the constraint
\[ \phi_{p_i}(\sigma_i, \sigma_{i+1}) \] (8)

**Fourth condition:** For each \( i \geq c \), the path between \( \sigma_i \) and \( \sigma_{i+1} \) must satisfy
\[ \bigwedge_{0 \leq j \leq i} \text{prop}(\phi_{k+1}^j) \]: Let \( p_i := \bigwedge_{0 \leq j \leq l} \text{prop}(\phi_{k+1}^j) \). We enforce this condition by putting the constraint
\[ \phi_{p_i}(\sigma_i, \sigma_{i+1}) \] (9)

**Fifth condition:** If \( r \) is a rule which is fired in the schedule between \( \sigma_c \) and \( \sigma_i \) such that \( r.u[x] > 0 \) for some shared variable \( x \), then for every rule \( r' \) which is fired in the schedule between \( \sigma_c \) and \( \sigma_i \), \( x \) should not appear in any of the fall guards of \( r' \): Recall that for every \( 0 \leq i < l \) and for every rule \( r \), the formula \( \phi_{p_i}(\sigma_i, \sigma_{i+1}) \) (as defined in \( \square \) and \( \square \)) has an existential variable \( \text{sum}^r_i \) which denotes the number of times the rule \( r \) has been fired in the schedule between \( \sigma_i \) and \( \sigma_{i+1} \). Hence to enforce this condition, we put the following constraint:

\[
\bigwedge_{r, r' \in R, x \in V} \left( \left( \bigvee_{c \leq i < l} \text{sum}^r_i > 0 \right) \land r.u[x] > 0 \land r' \in \text{Fall}_x \right) \Rightarrow \left( \bigwedge_{c \leq i < l} \text{sum}^r_i = 0 \right)
\]

Here \( \text{Fall}_x \) denote the set of all rules which have a fall guard with the variable \( x \).

Let \( \phi_{live} \) be the above formula. From the construction of the formula and Lemma \( \square \) it is clear that if \( (< Gr, P) \) is a witness then \( \phi_{live} \) is satisfiable. On the other hand, suppose \( \phi_{live} \) is satisfiable. We therefore get a sequence of configurations \( \eta_0, \ldots, \eta_l \) satisfying the formula \( \phi_{live} \). By Lemma \( \square \) we get that for every \( i \), there is a path \( P_i \) between \( \eta_i \) and \( \eta_{i+1} \) such that \( P_i(r) = \text{sum}^r_i \) for every rule \( r \) and the lifted path \( \text{lift}(P_i) \) satisfies the proposition \( p_i \). It is then easy to see that the pair \( (< Gr, Q) \) where \( Q \) is given by,
lift(P₀), lift(P₁),..., lift(P_{l-1})

is a witness.

It then follows that model-checking formulas in ELTL_{FT} is in NP.

D  Detailed proofs of Section 6

Proof of theorem 7

Proof. Given a sketch threshold automaton TA, formula \( \varphi \), and polynomial \( p \), we first guess in polynomial time an assignment \( \mu \). Clearly we can check if \( \mu \) makes the environment multiplicative in NP as the conditions of definition 2 can be encoded as an (integer) linear program. Using Theorem 6 we can decide if all executions of \( TA[\mu] \) satisfy \( \neg \varphi \) in co-non-deterministic polynomial time in \( TA[\mu] \). So we get a \( \Sigma^p_2 \) procedure for the bounded synthesis problem.

Proof of theorem 8

The upper bound follows from Theorem 4 and so we only prove the lower bound here, by giving a reduction from the \( \Sigma^p_2 \)-SAT problem, which is well-known to be \( \Sigma^p_2 \)-hard.

Let \( \exists x_1,\ldots,x_m \land \forall y_1,\ldots, y_k \phi(x,y) \) be a \( \Sigma^p_2 \)-3-SAT formula where \( \phi(x,y) \) is in disjunctive normal form. We construct a threshold automaton as follows: (See Figure D for an illustrative example)

Let Env = \((II, RC, N)\) be the environment where \( II \) consists of a single variable \( n \), \( RC \) is simply \( n \geq 1 \) and \( N \) is the identity function.

The sketch threshold automaton TA is constructed as follows: For every variable \( x_i \) we will have a location \( \ell_{x_i} \). Further we will also have a start location \( \ell_{x_0} \). Let \( a, b_1,\ldots,b_m, \bar{b}_1,\ldots,\bar{b}_m \) be shared variables. Our construction will ensure that \( a \) will never be incremented. Let \( v_1,\ldots,v_m \) be indeterminates which will appear in the threshold guards. Between \( \ell_{x_{i-1}} \) and \( \ell_{x_i} \) there are two rules of the form \((\ell_{x_{i-1}}, \ell_{x_i}, a < v_i \cdot n, \bar{b}_i++) \) and \((\ell_{x_{i-1}}, \ell_{x_i}, a \geq v_i \cdot n, b_i++) \). Since \( a \) will never be incremented (and hence always stay 0), it follows that depending on the value that \( v_i \) is assigned to, exactly one of these two rules can be fired.

For every variable \( y_i \) we will have a location \( \ell_{y_i} \) and two other locations \( \ell_{z_i}, \ell_{\bar{z}_i} \). Further we will also have a location \( \ell_{y_0} \) and a transition \((\ell_{x_m}, \ell_{y_0}, true, 0)\). Let \( c_1,\ldots,c_k, \bar{c}_1,\ldots,\bar{c}_k \) be shared variables. From \( \ell_{y_{i-1}} \) there are two rules, \((\ell_{y_{i-1}}, \ell_{z_i}, true, c_i++) \) and \((\ell_{y_{i-1}}, \ell_{\bar{z}_i}, true, \bar{c}_i++) \). Intuitively, firing the first rule corresponds to making \( y_i \) to be true and firing the second rule corresponds to making it false. We will ensure that if two different processes fire two different rules from \( \ell_{y_{i-1}} \) then all the processes get stuck: We add two rules \((\ell_{z_i}, \ell_{y_i}, c_i \geq n, 0) \) and \((\ell_{\bar{z}_i}, \ell_{y_i}, \bar{c}_i \geq n, 0) \). These two rules ensure that if two different processes fired two different rules from the location \( \ell_{y_{i-1}} \) then they all get stuck and cannot progress to \( \ell_{y_i} \).
For a variable \( x_i \), let \( \text{var}(x_i) = b_i \) and \( \text{var}(\bar{x}_i) = \bar{b}_i \). Similarly for a variable \( y_i \), let \( \text{var}(y_i) = c_i \) and \( \text{var}(\bar{y}_i) = \bar{c}_i \). Now for every disjunct \( D = p \land q \land r \) that appears in \( \phi(x,y) \) we have the following transition from \( \ell_{y_k} \) to a new location \( \ell_F \): \( (\ell_{y_k}, \ell_F, \text{var}(p)) \geq 1 \land \text{var}(q) \geq 1 \land \text{var}(r) \geq 1, 0 \). Finally both \( \ell_{y_k} \) and \( \ell_F \) have self-loops with no guards.

Let the above sketch threshold automaton be \( \text{TA} \). Let \( \varphi' \) be the following formula:

\[
\left( \bigvee_{(p \land q \land r) \in \phi(x,y)} \text{var}(p) \geq 1 \land \text{var}(q) \geq 1 \land \text{var}(r) \geq 1 \right)
\]

and let \( \varphi \) be the following formula: \((F G (\varphi' \implies \ell_{y_k} = 0)) \land (G \ell_F = 0)\). The first part of \( \varphi \) says that if any one of the rules between \( \ell_{y_k} \) and \( \ell_F \) get unlocked at some point, then all the processes leave \( \ell_{y_k} \) (and subsequently go to \( \ell_F \)). The second part of \( \varphi \) says that no process ever reaches \( \ell_F \). Hence \( \varphi \) can be satisfied by a path iff there is a process in \( \ell_{y_k} \) but none of the outgoing rules from \( \ell_{y_k} \) are unlocked.

We notice the following observation about the constructed threshold automaton: Let \( \mu \) be any assignment to the indeterminates. In any infinite run of \( \text{TA}[\mu] \) at least one process reaches the location \( \ell_{y_k} \). Further, in any run in which some process reaches the location \( \ell_{y_k} \), the run satisfies the following: For every \( 1 \leq i \leq m \), either all configurations of the run have \( b_i = 0 \) or all configurations of the run have \( \bar{b}_i = 0 \). Similarly for every \( 1 \leq j \leq k \), either all configurations of the run have \( c_i = 0 \) or all configurations of the run have \( \bar{c}_i = 0 \).

Let \( p \) be the identity function. We now show that there exists an assignment to the indeterminates \( \{v_i\} \) (using \( p(\text{TA}[\mu]) \) many bits) such that \( \neg\varphi \) is satisfied by every infinite run iff the formula \( \exists x_1, \ldots, x_m \forall y_1, \ldots, y_n \phi(x,y) \) is true.

Suppose there exists an assignment \( \mu \) to the indeterminates \( \{v_i\} \) such that \( \neg\varphi \) is true for every infinite run starting from every initial configuration in the threshold automaton \( \text{TA}[\mu] \). Consider the following assignment \( \nu \) for the variables \( x_i \): \( \nu(x_i) = \text{true} \) if \( \mu(v_i) > 0 \) and \( \nu(x_i) = \text{false} \) if \( \mu(v_i) = 0 \). It is now an easy check that the formula \( \forall y_1 \ldots y_n \phi(\nu(x), y) \) is true.

Suppose the formula \( \exists x_1, \ldots, x_m \forall y_1, \ldots, y_n \phi(x,y) \) is true. Let \( \nu \) be an assignment to the variables \( \{x_i\} \) such that \( \forall y_1 \ldots y_n \phi(\nu(x), y) \) is true. Consider an assignment \( \mu \) to the indeterminates \( \{v_i\} \) as follows: \( \mu(v_i) = 1 \) if \( \nu(x_i) = \text{true} \) and \( \mu(v_i) = 0 \) if \( \nu(x_i) = \text{false} \). It is once again easy to check that \( \neg\varphi \) is true for every infinite run starting from every initial configuration in the threshold automaton \( \text{TA}[\mu] \).
Fig. 6. Threshold automaton corresponding to the formula $\exists x_1, x_2 \forall y_1 (x_1 \land y_1 \land \bar{y}_1) \lor (\bar{x}_1 \land \bar{x}_2 \land y_1)$