Time-Bounded Resilience

Tajana Ban Kirigin[1], Jesse Comer[2], Max Kanovich[3], Andre Scedrov[2], and Carolyn Talcott[4]

[1] Faculty of Mathematics, University of Rijeka, HR bank@math.uniri.hr
[2] University of Pennsylvania, Philadelphia, USA
scedrov@math.upenn.edu, jacomer@seas.upenn.edu
[3] University College, London, UK m.kanovich@ucl.ac.uk
[4] SRI International, USA carolyn.talcott@sri.com

Abstract. Most research on system design has focused on optimizing various measures of *efficiency*. However, insufficient attention has been given to the design of systems optimizing *resilience*, the ability of systems to adapt to unexpected changes or adversarial disruptions. In our prior work, we formalized the intuitive notion of resilience as a property of cyber-physical systems by using a multiset rewriting language with explicit time. In the present paper, we study the computational complexity of a formalization of *time-bounded* resilience problems for the class of η*-simple* progressing planning scenarios, where, intuitively, it is simple to check that a system configuration is critical, and only a bounded number of rules can be applied in a single time step. We show that, in the time-bounded model with n (adversarially-chosen) disruptions, the corresponding time-bounded resilience problem for this class of systems is complete for the $\Sigma^{\mathsf{P}}_{2n+1}$ class of the polynomial hierarchy, PH. To support the formal models and complexity results, we perform automated experiments for time-bounded verification using the rewriting logic tool Maude.

1 Introduction

Resilience is "the ability of a system to adapt and respond to change (both environmental and internal)" [7]. In recent years, the task of formally defining and analyzing this intuitive notion has drawn interest across domains in computer science, ranging from systems engineering [28,31], particularly cyber-physical systems (CPS) [6,26], to artificial intelligence [33,14,35,17], programming languages [12,20], algorithm design [15,10], and more. Our previous work in [1] was particularly inspired by Vardi's paper [38], in which he articulated a need for computer scientists to reckon with the trade-off between efficiency and resilience.

In [1], we formalized resilience as a property of timed multiset rewriting (MSR) systems [24,22], which are suitable for the specification and verification of various goal-oriented systems. Although the related verification problems are undecidable in general, it was shown that these problems are PSPACE-complete for the class of *balanced* systems, in which facts are of bounded size, and rewrite rules do not change configuration size. A primary challenge in [1] was the formalization of the disruptions against which systems must be resilient. This was achieved by separating the system from the environment, delineating between rules applied by the system and those imposed on the system, such as changes in conditions, regulations, or mission objectives.

Main Contributions. This paper formalizes the notion of *time-bounded* resilience. We focus on the class of η-simple progressing planning scenarios (PPS) and investigate the computational complexity of the corresponding verification problem. Time-bounded resilience is motivated by bounded model checking and automated experiments, which can help system designers verify properties and find counterexamples where their specifications do not satisfy time-bounded resilience. Moreover, bounded versions of resilience problems arise naturally when the missions of the systems being modeled are necessarily bounded at some level. The main contributions of the paper are as follows.

1. We define time-bounded resilience as a property of planning scenarios. Intuitively, a resilient system can accomplish its mission within the given time bounds, even in the presence of a bounded number of disruptions (cf. Definition 11).
2. We investigate the computational complexity of time-bounded resilience problems, showing that for the class of η-simple PPSs with facts of bounded size [23], the time-bounded resilience problem with n updates is complete for the $\Sigma^{\mathsf{P}}_{2n+1}$ class of the polynomial hierarchy, PH (Corollary 1).
3. We demonstrate that our formalization can be automated, using the rewriting logic tool Maude to perform experiments verifying time-bounded resilience (Section 5).

Expository Example. In [1], our study of resilience was motivated by current research into CPSs that perform complex, safety-critical tasks in hostile and unpredictable environments, often autonomously. In this paper, we expand our perspective to consider resilience properties of a broad class of multi-agent systems. For expository purposes, we utilize a running example of a researcher planning travel to attend and present research at a conference. The system rules represent actions of the researcher, while update rules represent travel disruptions and changes to the conference organization. Ultimately, the travel planning process is pointless if the researcher does not arrive at his destination in time for the main event. Consequently, the researcher desires to establish a *resilient* plan, which will allow him to accomplish his goal in spite of some bounded number of disruptions. Details of this planning scenario will be developed throughout Section 2, and our Maude implementation in Section 5 will be used to analyze its resilience.

Outline. Section 2 reviews the timed MSR framework used in Section 3 to define time-bounded resilience. In Section 4, we investigate the complexity of the verification problem. Section 5 showcases our results on automated verification obtained using Maude. In Section 6, we conclude with a discussion of related and future work.

2 Multiset Rewriting Systems

In this section, we review the framework of timed MSR models introduced in our previous work [21,23,24].

2.1 The Rewriting Framework

Terms and Formulas. We fix a finite first-order alphabet Σ with constant, function, and predicate symbols, together with a finite set $\mathcal{B}$ of *base types*. Each constant is associated with a unique base type, and we write Σ_{Cons} to denote the set of all constants in

Σ. Each predicate symbol R (resp. function symbol f) is associated with a unique *tuple type* (resp. *arrow type*) $b_1 \times \ldots \times b_k$ (resp. $b_1 \times \ldots \times b_k \to b$), where $b_1, \ldots, b_k, b \in \mathcal{B}$ and k is the arity of R (resp. f). We also assume that Σ contains a special predicate symbol Time with arity zero (more on this later).

We fix sets $\mathcal{V}_{\text{FO}}$ of (first-order) *variables* and $\mathcal{G}$ of *ground constants*, disjoint from each other and from Σ, where each element in $\mathcal{V}_{\text{FO}} \cup \mathcal{G}$ has an associated base type in $\mathcal{B}$. We further assume that $\mathcal{V}_{\text{FO}}$ and $\mathcal{G}$ each contain countably infinitely-many elements associated to each base type. These ground constants will be used later on to instantiate variables "created" by rewrite rules. *Terms* over Σ are constructed according to the grammar

$$t := x \mid c \mid f(t_1, \ldots, t_k),$$

where x is in $\mathcal{V}_{\text{FO}}$, c is in Σ_{Cons}, f is a function symbol of type $b_1 \times \ldots \times b_k \to b$, and each t_i is a term of type b_i for $i \leq k$ (in which case $f(t_1, \ldots, t_k)$ is a term of type b). *Ground terms* over Σ are constructed similarly:

$$a := d \mid c \mid f(a_1, \ldots, a_k),$$

where d is in $\mathcal{G}$, c is in Σ_{Cons}, f is a function symbol of type $b_1 \times \ldots \times b_k \to b$, and each a_i is a ground term of type b_i for $i \leq k$ (in which case $f(a_1, \ldots, a_k)$ is a ground term of type b). We write $\mathcal{G}_{\text{Terms}}$ for the collection of ground terms over Σ. If R is a predicate symbol of type $b_1 \times \ldots \times b_k$ and $t_1, \ldots, t_k$ are terms of type $b_1, \ldots, b_k$, respectively, then $R(t_1, \ldots, t_k)$ is an *atomic formula*. Similarly, if $a_1, \ldots, a_k$ are ground terms of type $b_1, \ldots, b_k$, respectively, then $R(a_1, \ldots, a_k)$ is an *atomic fact* (or just *fact*).

Modeling Discrete Time. We fix a countably infinite set $\mathcal{V}_{\mathsf{Time}} = \{T_i \mid i \in \mathbb{N}\}$ of *time variables*. *Timestamped atomic formulas* are of the form $F@(T + D)$, where F is an atomic formula, T is a time variable, and D is a natural number; note that if $D = 0$, we prefer to write $F@T$ instead of $F@(T + 0)$. *Timestamped facts* are of the form $F@t$, where F is an atomic fact and $t \in \mathbb{N}$ is its *timestamp*. For brevity, we frequently refer to timestamped facts simply as facts. Clearly, given a timestamped atomic formula $F@(T + D)$, we can obtain a timestamped fact $G@t$ by setting G to be the result of uniformly substituting ground terms for the variables in F and setting $t = N + D$.

Configurations and Rewrite Rules. *Configurations* are multisets of timestamped facts, $\mathcal{S} = \{\mathsf{Time}@t, F_1@t_1, \ldots, F_n@t_n\}$, with a single occurrence of a Time fact, whose timestamp is the *global time* in $\mathcal{S}$. We write $\text{Values}(\mathcal{S})$ to denote the set of all ground terms and timestamps occurring in $\mathcal{S}$. Configurations are modified by *multiset rewrite rules*. Only one rule, Tick, modifies global time:

$$\mathsf{Time}@T \longrightarrow \mathsf{Time}@(T + 1) \tag{1}$$

where T is a time variable. The Tick rule modifies a configuration to which it is applied by advancing the global time by one. The remaining rules are *instantaneous* in that they do not modify the global time but may modify the remaining facts of a configuration. Instantaneous rules are given by expressions of the form

$$\begin{aligned} &\mathsf{Time}@T, \mathcal{W}, F_1@T_1, \ldots F_n@T_n \mid \mathcal{C} \\ &\quad \longrightarrow \mathsf{Time}@T, \mathcal{W}, Q_1@(T + D_1), \ldots Q_m@(T + D_m) \end{aligned} \tag{2}$$

where $\mathcal{W}$ (the *side condition* of the rule) is a multiset of timestamped atomic formulas, $F_i@T_i$ is a timestamped atomic formula for each $i \leq n$, and $Q_j@(T + D_j)$ is a timestamped atomic formula for each $j \leq m$. The *precondition* of the rule is the multiset $\{\mathsf{Time}@T\} \cup \mathcal{W} \cup \{F_i@T_i \mid i \leq n\}$, while its *postcondition* is the multiset $\{\mathsf{Time}@T\} \cup \mathcal{W} \cup \{Q_j@(T + D_j) \mid j \leq m\}$. We require that no atomic formula in the multiset $\{F_i@T_i \mid i \leq n\}$ appears with the same multiplicity as it appears in the multiset $\{Q_j@(T + D_j) \mid j \leq m\}$. Furthermore, no timestamped atomic formulas containing the predicate Time can occur in $\{F_i@T_i \mid i \leq n\} \cup \{Q_j@(T + D_j) \mid j \leq m\}$. The *guard* $\mathcal{C}$ of the rule is a set of *time constraints* of the form

$$T_1 > T_2 \pm N \quad \text{or} \quad T_1 = T_2 \pm N,$$

where T_1 and T_2 are time variables and $N \in \mathbb{N}$ is a natural number; all constraints in $\mathcal{C}$ must involve only the time variables occurring in the rule's precondition.

A *ground substitution* is a partial map $\sigma : \mathcal{V}_{\text{FO}} \cup \mathcal{V}_{\mathsf{Time}} \to \mathcal{G}_{\text{Terms}} \cup \mathbb{N}$ which maps first-order variables to ground terms and time variables to natural numbers. Given a multiset W of timestamped atomic formulas, we write $W\sigma$ to denote the multiset of timestamped facts obtained by simultaneously substituting all first-order variables and time variables in W with their image under σ. Given a set $\mathcal{C}$ of time constraints with time variables from W, we say that $\mathcal{C}\sigma$ is *satisfied* if each time constraint in $\mathcal{C}$ evaluates to true for the substituted timestamps. Given a multiset W of timestamped atomic formulas, we write $\text{Var}(W)$ to denote the set of first-order variables and time variables occurring in W. Given an instantaneous rule r given by $W \mid \mathcal{C} \longrightarrow W'$, we write $\text{Fresh}(r)$ to denote the set $\text{Var}(W') \setminus \text{Var}(W)$.

A ground substitution matching an instantaneous rule r given by $W \mid \mathcal{C} \longrightarrow W'$ to a configuration $\mathcal{S}$ is a ground substitution σ with $\text{dom}(\sigma) = \text{Var}(W \cup W')$ such that every element of $\text{Var}(W)$ is matched to an element in $\text{Values}(\mathcal{S})$, and the restriction of σ to $\text{Fresh}(r)$ is an injective map whose range is contained in $\mathcal{G} \setminus \text{Values}(\mathcal{S})$. In other words, σ assigns first-order variables (resp. time variables) occurring in W to ground terms (resp. timestamps) occurring in $\mathcal{S}$, and each distinct first-order variable in $\text{Fresh}(r)$ to a *fresh* ground constant which does not occur in $\mathcal{S}$.

An instantaneous rule r given by $W \mid \mathcal{C} \longrightarrow W'$ is *applicable* to a configuration $\mathcal{S}$ if there exists a ground substitution matching r to $\mathcal{S}$ such that $W\sigma \subseteq \mathcal{S}$ and $\mathcal{C}\sigma$ is satisfied; in this case, we refer to the expression $r\sigma$ given by $W\sigma \mid \mathcal{C}\sigma \longrightarrow W'\sigma$ as a *instance* of the rule r. The result of applying the rule instance $r\sigma$ to $\mathcal{S}$ is the configuration $(\mathcal{S} \setminus W\sigma) \cup W'\sigma$. If $\mathcal{W}$ is the side condition of r, and T is the global time in $\mathcal{S}$, then we say that the timestamped facts occurring in $(W \setminus (\mathcal{W} \cup \{\mathsf{Time}@T\}))\sigma$ are *consumed*, while those in $(W' \setminus (\mathcal{W} \cup \{\mathsf{Time}@T\}))\sigma$ are *created*. Note that a fact for the predicate Time is never created by an instantaneous rule. We write $\mathcal{S} \longrightarrow_r \mathcal{S}'$ for the one-step relation where the configuration $\mathcal{S}$ is rewritten to $\mathcal{S}'$ using an instance of the rule r. It is worth emphasizing, at this point, that configurations are *grounded*, while rewrite rules are *symbolic*.

Some examples. We now give some examples to elucidate our formalism. Consider the alphabet containing the predicate symbols Time, At, $\underline{\mathsf{Event}}$, $\underline{\mathsf{Attended}}$, and Flight_D (where $D \in \{1, \ldots, 12\}$), and the constant symbols no, done, main, $\mathsf{airport}$, center,

id_{14}, and id_{215}. Recall that, in our expository example, we are modeling a researcher with a goal of traveling to attend a conference. We interpret the timestamped atomic formula $\mathsf{Flight}_D(id, c_1, c_2)@T$ to mean that the flight with flight id id from city c_1 to city c_2 departs at time T and has a duration of approximately D hours.

The timestamped fact $\mathsf{At}(\mathsf{FRA}, \mathsf{center})@0$ is interpreted to mean that the researcher is at Frankfurt city center at the initial time step 0. For this scenario, each time step is interpreted as the passage of one minute. For ease of readability, we adopt a more convenient representation of timestamps, with 0 denoting midnight on the initial day of the planning scenario. Then, we write $\mathsf{Time}@(3d\ 14{:}42)$ to indicate that the current time is 14:42 on the 3rd day of the scenario. We do this is in lieu of writing the more burdensome timestamp $\mathsf{Time}@5202$. The fact $\underline{\mathsf{Event}}(\mathsf{main}, \mathsf{id}_{215})@(5d\ 12{:}00)$ specifies that the main event of the conference, with event identifier 215, will take place at noon on the 5th day. Bringing this all together, consider the following configuration.

$$\begin{array}{l}\{\mathsf{Time}@(3d\ 14{:}42), \underline{\mathsf{Attended}}(\mathsf{main}, \mathsf{no})@0,\ \mathsf{At}(\mathsf{FRA}, \mathsf{airport})@(3d\ 14{:}05),\\ \underline{\mathsf{Event}}(\mathsf{main}, \mathsf{id}_{215})@(5d\ 12{:}00), \mathsf{Flight}_2(\mathsf{id}_{14}, \mathsf{FRA}, \mathsf{DBV})@(3d\ 15{:}25)\}\end{array} \tag{3}$$

This configuration describes a state of the system. The time is 14:42 on the 3rd day of the scenario, and the researcher arrived at Frankfurt airport (FRA) at 14:05. The main event of the conference is at noon in two days in Dubrovnik (DBV), and has, obviously, not yet been attended by the researcher. Flight id_{14} is a direct flight from Frankfurt to Dubrovnik, which departs at 15:25 and has a duration of approximately two hours.

In addition to modeling states of the system via configurations, we also want our formalism to be able to model actions taken by the researcher, such as boarding a given flight. To this end, consider the rule

$$\begin{array}{r}\mathsf{Time}@T, \mathsf{Flight}_2(a, x, y)@T_1, \mathsf{At}(x, \mathsf{airport})@T_2, \mid T = T_1, T_2 + 30 \leq T\\ \longrightarrow \mathsf{Time}@T, \mathsf{Flight}_2(a, x, y)@T_1, \mathsf{At}(y, \mathsf{airport})@(T + 120),\end{array} \tag{4}$$

with side condition $\{\mathsf{Flight}_2(a, x, y)@T_1\}$. This rule means that if the departure time of a two-hour flight with flight id a from city x to city y will depart at time T, and the researcher is at the airport in city x at time T_2, where T_2 is at least 30 minutes prior to T, then he can take the flight, arriving at the airport in city y after two hours.

Note that the rule (Eq. 4) is not applicable to the configuration (Eq. 3). In particular, the time constraint $T = T_1$ cannot be satisfied by any ground assignment for the rule to the configuration. However, rule (Eq. 4) *is* applicable to the configuration resulting from the successive application of 43 Tick rules to configuration (Eq. 3), which results in the same configuration, except with the timestamp for Time updated to $(3d\ 15{:}25)$ (i.e., the departure time of the flight). Then the ground substitution σ given by

$$\begin{array}{ll}\sigma(T) = 3d\ 15{:}25 & \sigma(a) = \mathsf{id}_{14}\\ \sigma(T_1) = 3d\ 15{:}25 & \sigma(x) = \mathsf{FRA}\\ \sigma(T_2) = 3d\ 14{:}05 & \sigma(y) = \mathsf{DBV}\end{array}$$

applied to the rule (Eq. 4) yields an instance which can be applied to configuration (Eq. 3), resulting in the following configuration:

$$\begin{array}{l}\{\mathsf{Time}@(3d\ 15{:}25), \underline{\mathsf{Attended}}(\mathsf{main}, \mathsf{no})@0,\ \mathsf{At}(\mathsf{DBV}, \mathsf{airport})@(3d\ 17{:}25),\\ \underline{\mathsf{Event}}(\mathsf{main}, \mathsf{id}_{215})@(5d\ 12{:}00), \mathsf{Flight}_2(\mathsf{id}_{14}, \mathsf{FRA}, \mathsf{DBV})@(3d\ 15{:}25)\}.\end{array} \tag{5}$$

Timed MSR Systems. We now turn to the timed MSR systems introduced in [24].

Definition 1. *A* timed MSR system $\mathcal{A}$ *is a set of rules containing only instantaneous rules (Eq. 2) and the* Tick *rule (Eq. 1).*

A sequence of consecutive rule applications represents an execution or process within the system. A *trace* of timed MSR rules $\mathcal{A}$ starting from an initial configuration $\mathcal{S}_0$ is a sequence of configurations: $\mathcal{S}_0 \longrightarrow \mathcal{S}_1 \longrightarrow \mathcal{S}_2 \longrightarrow \cdots \longrightarrow \mathcal{S}_n$, such that for all $0 \leq i \leq n-1$, $\mathcal{S}_i \longrightarrow_{r_i} \mathcal{S}_{i+1}$ for some $r_i \in \mathcal{A}$. For our complexity results, we assume traces are annotated with the rule instances used to obtain the next configuration in the trace, so valid traces can be recognized in polynomial time (cf. Remark 4).

Reachability problems for MSR systems are reduced to the existence of traces over given rules from some initial configuration to some specified configuration. Since reachability problems are undecidable in general [24], some restrictions are imposed in order to obtain decidability[5]. In particular, we use MSR systems with only *balanced* rules.

Definition 2 (Balanced Rules, [24]). *A timed MSR rule is* balanced *if the numbers of facts on left and right sides of the rule are equal.*

Systems containing only balanced rules represent an important class of *balanced systems*, for which several variants of the reachability problem have been shown to be decidable [23]. Balanced systems are suitable, *e.g.*, for modeling scenarios with a fixed amount of total memory. Balanced systems have the following important property:

Proposition 1 ([23]). *Let $\mathcal{R}$ be a set of balanced rules. Let $\mathcal{S}_0$ be a configuration with exactly m facts (counting multiplicities). Let $\mathcal{S}_0 \longrightarrow \cdots \longrightarrow \mathcal{S}_n$ be an arbitrary trace of $\mathcal{R}$ rules starting from $\mathcal{S}_0$. Then for all $0 \leq i \leq n$, $\mathcal{S}_i$ has exactly m facts.*

2.2 Progressing Timed Systems

In this section, we review a particular class of timed MSR systems, called *progressing timed MSR systems* (PTSs) [21,22], in which only a bounded number of rules can be applied in a single time step. This is a natural condition, similar to the *finite-variability assumption* used in the temporal logic and timed automata literature [18].

Definition 3 (Progressing Timed System, [21]). *An instantaneous rule r of the form in (Eq. 2) is* progressing *if the following all hold: i) $n = m$ (i.e., r is balanced); ii) r consumes* only *facts with timestamps in the past or at the current time,* i.e.*, in (Eq. 2), the set of constraints $\mathcal{C}$ of r contains the set $\mathcal{C}_r = \{\ T \geq T_i \mid F_i@T_i,\ 1 \leq i \leq n\ \}$; iii) r creates* at least one *fact with timestamp greater than the global time,* i.e.*, in (Eq. 2), $D_i \geq 1$ for at least one $i \in \{1, \ldots, n\}$. A timed MSR system $\mathcal{A}$ is a* progressing timed MSR system (PTS) *if all instantaneous rules of $\mathcal{A}$ are progressing.*

Note that the rule (Eq. 4) is progressing. A timestamped fact in a configuration $\mathcal{S}$ is a *future fact* if its timestamp is strictly greater than the timestamp of the $\mathsf{Time}@T$ fact in $\mathcal{S}$. Future facts are "not available" in the sense that they cannot be consumed by a progressing rule before a sufficient number of Tick rule applications.

Remark 1. For readability, we assume the set of constraints for all rules r, contains the set $\mathcal{C}_r$, as in Definition 3, and do not always write $\mathcal{C}_r$ explicitly.

[5] For a discussion of various conditions in the model that may affect complexity, see [23,24].

2.3 Timed MSR for the specification of resilient systems

We now review additional notation for the purpose of specifying resilience, as introduced in [1]. The resilience framework divides the system from an external entity, such as the environment, regulatory authorities, or an adversary. We model various types of disruptive changes to the system state or goals.

Definition 4 (Planning Configuration, [1]). *Let $\Sigma_P = \Sigma_G \uplus \Sigma_C \uplus \Sigma_S \uplus \{\mathsf{Time}\}$ consist of four pairwise disjoint sets of predicate symbols, Σ_G, Σ_C, Σ_S and $\{\mathsf{Time}\}$. Facts constructed using predicates from Σ_G are called* goal facts*, from Σ_C* critical facts*, and from Σ_S* system facts*. Facts constructed using predicates from $\Sigma_C \cup \Sigma_G$ are called* planning facts*. Configurations over Σ_P predicates are called* planning configurations*.*

For readability, we underline predicates in planning facts and refer to planning configurations as configurations for short. The behavior of the system is represented by traces of MSR rules. A system should achieve its goal while not violating predetermined critical conditions. This is made precise in the following two definitions.

Definition 5 (Critical/Goal Configurations, [1]). *A* critical (resp. goal) configuration specification *$\mathcal{CS}$ (resp. $\mathcal{GS}$) is a set of pairs $\{\langle \mathcal{S}_1, \mathcal{C}_1\rangle, \ldots, \langle \mathcal{S}_n, \mathcal{C}_n\rangle\}$, with each pair $\langle \mathcal{S}_j, \mathcal{C}_j\rangle$ being of the form $\langle \{F_1@T_1, \ldots, F_{p_j}@T_{p_j}\}, \mathcal{C}_j\rangle$, where $T_1, \ldots, T_{p_j}$ are time variables, $W = \{F_1, \ldots, F_{p_j}\}$ is a multiset of timestamped atomic formulas, with at least one occurrence of a critical (resp. goal) predicate symbol, and $\mathcal{C}_j$ is a set of time constraints involving only variables $T_1, \ldots, T_{p_j}$. A configuration $\mathcal{S}$ is a* critical configuration *w.r.t. $\mathcal{CS}$ (resp. a* goal configuration *w.r.t. $\mathcal{GS}$) if for some $1 \leq i \leq n$, there is a grounding substitution σ with $dom(\sigma) = Var(W)$ such that $\mathcal{S}_i\sigma \subseteq \mathcal{S}$ and $\mathcal{C}_i\sigma$ is satisfied.*

Definition 6 (Compliant Traces, [1]). *A trace is* compliant *with respect to a critical configuration specification $\mathcal{CS}$ if it does not contain any critical configuration w.r.t. $\mathcal{CS}$.*

Note that critical configuration specifications and goal configuration specifications, like rewrite rules, are *symbolic*. Reaching a critical configuration may be interpreted as a *safety violation*, while a compliant trace may be interpreted as a *safe trace*. As an example, suppose that in the example alphabet introduced earlier, the predicate symbol $\underline{\mathsf{Attended}}$ is in Σ_C, while the predicate symbol $\underline{\mathsf{Event}}$ is in Σ_G. Then the goal configuration specification

$$\{\langle \{\underline{\mathsf{Attended}}(\mathsf{main}, \mathsf{done})@T_1, \underline{\mathsf{Event}}(\mathsf{main}, x)@T_2\}, \emptyset\rangle\}$$

indicates that the main event should be attended, while the critical configuration specification

$$\{\langle\, \mathsf{Time}@T, \underline{\mathsf{Attended}}(\mathsf{main}, \mathsf{no})@T_1, \underline{\mathsf{Event}}(\mathsf{main}, x)@T_2\}, \{T > T_2\}\rangle\}$$

denotes that it is critical not to participate in the main event.

Definition 7 (System Rules and Update Rules, [1]). *Fix a planning alphabet Σ_P. A* system rule *is either the* Tick *rule (Eq. 1) or a rule of form in (Eq. 2) which does not consume or create planning facts. An* update rule *is an instantaneous rule that is of one of the following types: (a) a* system update rule (SUR) *such that planning facts can only occur in the side condition of the rule; or (b) a* goal update rule (GUR) *that either consumes or creates at least one goal fact and such that critical facts can only occur in the side condition of the rule.*

For example, the following system rule specifies that the traveler needs 40 minutes to get from the departing city center to the airport:

$$\begin{aligned} &\mathsf{Time}@T,\ \mathsf{At}(x, \mathsf{center})@T_1 \mid T_1 \leq T \\ &\quad \longrightarrow\ \mathsf{Time}@T, \mathsf{At}(x, \mathsf{airport})@(T + 40). \end{aligned}$$

The rule (Eq. 4) is another example of a system rule. System rules specify the behavior of the system, while disruptions are modeled via update rules. Intuitively, GUR model external interventions in the system, such as mission changes, additional tasks, etc., while SUR model changes in the system that are not due to the intentions of the system's agents, *e.g.*, technical errors or malfunctions such as flight delays. Both goal and system update rules can create and/or consume system facts, which technically simplifies modeling the impact of changes on the system and its response. For example, the following GUR models a change in the scheduled time of the main event.

$$\mathsf{Time}@T,\ \underline{\mathsf{Event}}(\mathsf{main}, x)@T_1, \longrightarrow \mathsf{Time}@T, \underline{\mathsf{Event}}(\mathsf{main}, x)@(T + 60),$$

while the following SUR models a 30-minute flight delay:

$$\mathsf{Time}@T, \mathsf{Flight}_D(a, x, y)@T_1 \longrightarrow \mathsf{Time}@T, \mathsf{Flight}_D(a, x, y)@(T + 30).$$

Definition 8 (Planning Scenario, [1]). *If $\mathcal{R}$ and $\mathcal{E}$ are sets of system and update rules, $\mathcal{GS}$ and $\mathcal{CS}$ are a goal and critical configuration specifications, and $\mathcal{S}_0$ is an initial configuration, then the tuple $(\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}_0)$ is a* planning scenario.

Definition 9 (Progressing Planning Scenario (PPS)). *We say that a planning scenario $(\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}_0)$ is* progressing *if all rules in $\mathcal{R}$ and $\mathcal{E}$ are progressing.*

The progressing condition in Definition 9 bounds the number of rules that can be applied in a single unit of time (cf. Proposition 2). We also assume an upper-bound on the size of facts allowed to occur in traces, where the size of a timestamped fact $F@T$ is the number of symbols from Σ occurring in F, counting repetitions. Without this bound (among other restrictions), any interesting decision problem is undecidable [13,23]. We also confine attention to classes of η-*simple* PPSs, defined below.

Definition 10. *Let η denote a fixed positive integer. We say that a planning scenario $A = (\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}_0)$ is* η-simple *if the total number of variables (including both first-order and time variables) appearing in each pair $\langle \mathcal{S}_i, \mathcal{C}_i \rangle$ in $\mathcal{CS}$ is less than η.*

For every planning scenario $A = (\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}_0)$, there exists some least η such that A is η-simple; intuitively, this η is a measure of the complexity of verifying compliance of traces with respect to $\mathcal{CS}$. Proposition 4 in Section 4 makes this intuition precise.

Remark 2. By inspecting the rules and the critical configuration specification, it is easy to check that our expository travel example is 3-simple and progressing.

3 Time-bounded Resilience Verification Problems

In this section, we formalize time-bounded resilience as a property of planning scenarios. Intuitively, we want to capture the notion of a system which can achieve its goal within a fixed amount of time, despite the application of up to n instances of update rules. An initial idea might be to require that the system can achieve its goal in the allotted time, regardless of when updates are applied. However, this is too restrictive: many systems will fail to achieve their goal in the face of adversarial actions which can be applied at arbitrary times. Instead, the system will have $a + b$ time units to achieve its goal, and update rules can only be applied in the first a time steps; the last b time steps are the *recovery time* afforded to the system.

Definition 11 (The (n, a, b)-resilience problem). *Let $a \in \mathbb{Z}^+$ and $b \in \mathbb{N}$. We define (n, a, b)-resilience by recursion on n. Inputs to the problem are planning scenarios $A = (\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}_0)$. A trace is* $(0, a, b)$-resilient with respect to A *if it is a compliant trace of $\mathcal{R}$ rules from $\mathcal{S}_0$ to a goal configuration and contains at most $a+b$ applications of the* Tick *rule. For $n > 0$, a trace τ is* (n, a, b)-resilient with respect to A *if*

1. *τ is $(0, a, b)$-resilient with respect to A, and*
2. *for any system or goal update rule $r \in \mathcal{E}$ applied to a configuration $\mathcal{S}_i$ in τ, with $\mathcal{S}_i \longrightarrow_r \mathcal{S}'_{i+1}$, where global time t_i in $\mathcal{S}_i$ satisfies $d_i = t_i - t_0 \leq a$, there exists a* reaction trace *τ' of $\mathcal{R}$ rules from $\mathcal{S}'_{i+1}$ to a goal configuration $\mathcal{S}'$ such that τ' is $(n - 1, a - d_i, b)$-resilient with respect to $A' = (\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}'_{i+1})$.*

A planning scenario $A = (\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}_0)$ is (n, a, b)-resilient *if an (n, a, b)-resilient trace with respect to A exists. The (n, a, b)-resilience problem is to determine if a given planning scenario A is (n, a, b)-resilient.*

Figure 1 provides a visual depiction of Definition 11.

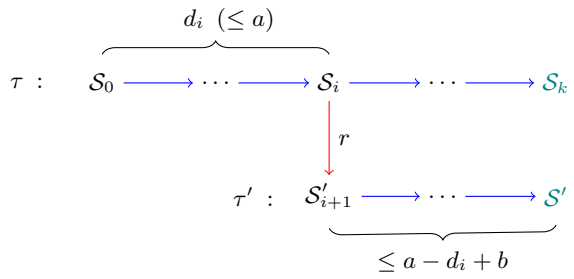

Fig. 1: An (n, a, b)-resilient trace τ and an $(n - 1, a - d_i, b)$-resilient reaction trace τ'. The horizontal arrows correspond to system rule applications, while the downward-facing arrow represents an update rule application. The configurations $\mathcal{S}_k$ and $\mathcal{S}'$ on the far right are goal configurations.

The reaction trace τ' in Definition 11 can be interpreted as a change in the plan τ, made in response to an external disruption (i.e., the system/goal update rule r) imposed

on the system. Note that it is this "replanning" aspect of our definition that intuitively distinguishes it from the related notion of robustness.

Remark 3. In Definition 11, the global time t' in $\mathcal{S}'$ satisfies $t' - t_0 \leq a + b$; *i.e.*, despite the application of n instances of update rules, an (n, a, b)-resilient trace reaches a goal within $a + b$ time units. Furthermore, observe that a trace is (n, a, b)-resilient with respect to a planning scenario A if and only if it is (n, a, b')-resilient with respect to A for all $b' \geq b$. Similarly, all (n, a, b)-resilient traces with respect to A are (n', a, b)-resilient with respect to A for all $n' \leq n$.

It is worthwhile to note that Definition 11 can be seen as a modification of [1, Definitions 9-10], in which (i) we include the parameters a and b, (ii) we consider both system and goal update rules simultaneously, and (iii) the *recoverability condition*, which is not mentioned in this work, is the total relation on configurations of A.

4 Computational Complexity of Time-Bounded Resilience

In this section, we state and prove our results on the computational complexity of the time-bounded resilience problem defined in Section 3. To see this, we first state a known bound on the number of instances of instantaneous rules appearing between two consecutive instances of $Tick$ rules in a trace of only progressing rules.

Proposition 2 ([21]). *Let $\mathcal{R}$ be a set of progressing rules, $\mathcal{S}_0$ an initial configuration and m the number of facts in $\mathcal{S}_0$. For all traces τ of $\mathcal{R}$ rules starting from $\mathcal{S}_0$, let*

$$\mathcal{S}_i \longrightarrow_{Tick} \mathcal{S}_{i+1} \longrightarrow \cdots \longrightarrow \mathcal{S}_j \longrightarrow_{Tick} \mathcal{S}_{j+1}$$

be any subtrace of τ with exactly two instances of the $Tick$ rule, one at the beginning and the other at the end. Then $j - i \leq m$.

Proposition 2 guarantees that the size (n, a, b)-resilient traces of a progressing planning scenario A are polynomially-bounded in the size of the input representation of A.

Proposition 3. *Let $A = (\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}_0)$ be a PPS and m be the number of facts in $\mathcal{S}_0$. Then the length of any (n, a, b)-resilient trace of A is bounded by $(a + b + 1)m$.*

In preparation for our (n, a, b)-resilience upper bound result (Theorem 1), we now turn to the complexity of some fundamental decision problems pertaining to planning scenarios. We only state the problems and their complexity for η-simple PPSs with facts of bounded size; more detail can be found in the technical report [3].

Definition 12. *The* goal (resp. critical) recognition problem *is to determine, given a planning scenario $A = (\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}_0)$ and a configuration $\mathcal{S}$, whether or not $\mathcal{S}$ is a goal (resp. critical) configuration w.r.t. $\mathcal{GS}$ (resp. $\mathcal{CS}$); cf. Definition 5.*

More broadly, we are interested in checking trace compliance.

Definition 13. *The* trace compliance problem *is to determine, given a planning scenario $A = (\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}_0)$ and a trace τ of $\mathcal{R}$-rules starting from $\mathcal{S}_0$, whether or not τ is compliant w.r.t. $\mathcal{CS}$ (cf. Definition 6).*

They key observation, and the one underlying our restriction to η-simple PPSs (cf. Definition 10), is that the trace compliance problem is tractable for this class.

Proposition 4. *For η-simple planning scenarios, the trace compliance problem is in* P.

Remark 4. If $A = (\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}_0)$ is an η-simple PPS, and $\mathcal{S}$ a configuration with the same number of facts as $\mathcal{S}_0$, then given an appropriate ground substitution, we can verify in polynomial time in the size of A that $\mathcal{S}$ is a goal configuration w.r.t. $\mathcal{GS}$. In the proof of Theorem 1, for ease of exposition, we will also assume that these ground substitutions come with a pointer to the appropriate pair $\langle \mathcal{S}_i, \mathcal{C}_i \rangle$ in $\mathcal{GS}$ to which the substitution should be applied. Furthermore, given an arbitrary PPS $A = (\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}_0)$, if $\mathcal{S}$ is a configuration with the same number of facts as $\mathcal{S}_0$, then given an appropriate ground substitution, we can verify in polynomial time in the size of A that $\mathcal{S}$ is a goal configuration w.r.t. $\mathcal{CS}$. Similarly, given an appropriate ground substitution, we can verify in polynomial time in the size of A that a rule r is applicable to $\mathcal{S}$, and whether or not $\mathcal{S}'$ is the result of this application.

We now turn our attention to the computational complexity of the (n, a, b)-resilience problem. To establish our complexity results, we will utilize the quantifier-alternation characterization of PH (cf. [2,37,34]), according to which a decision problem is in Σ_n^{P} (for n odd) if and only if there exists a polynomial-time algorithm M such that an input x is a *yes* instance of the problem if and only if

$$\exists u_1 \forall u_2 \exists u_3 \ldots \forall u_{n-1} \exists u_n \; M(x, u_1, \ldots, u_n) \text{ accepts},$$

where the u_i are polynomially-bounded in the size of x. We now establish an upper bound on the complexity of the (n, a, b)-resilience problem (cf. Definition 11).

Theorem 1. *For η-simple PPSs with traces containing only facts of bounded size and all $a \in \mathbb{Z}^+$ and $n, b \in \mathbb{N}$, there exists a decision procedure of complexity $\Sigma_{2n+1}^{\mathsf{P}}$ for the (n, a, b)-resilience problem.*

Proof. We show by induction on n that, for each $a \in \mathbb{Z}^+$ and $b \in \mathbb{N}$, there exists a polynomial-time algorithm $M_n^{a,b}$ such that an η-simple PPS $A = (\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}_0)$ is (n, a, b)-resilient if and only if

$$\exists \mathcal{T}_0 \forall \rho_1 \exists \mathcal{T}_1 \ldots \forall \rho_n \exists \mathcal{T}_n \; M_n^{a,b}(A, \tau_0, \tau_1, \ldots, \tau_n, \rho_1, \ldots, \rho_n) \text{ accepts}. \tag{6}$$

The existentially quantified variables $\mathcal{T}_i$ range over triples of the form (τ_i, σ_i, j_i), where τ_i is a trace of $\mathcal{R}$-rules and σ_i is a ground substitution from $\langle \mathcal{S}_{j_i}, C_{j_i} \rangle$ in $\mathcal{GS}$ to the last configuration of τ_i. By Proposition 3, such a witness $\mathcal{T}_i$ is polynomially-bounded in the size of the input representation of A. The universally quantified variables ρ_i range over triples of the form (r_i, σ_i, j_i), where $r_i \in \mathcal{E}$ and σ_i is a ground substitution from the j_i^{th} configuration of τ_{i-1} to the first configuration of τ_i. The witnesses ρ_i are also clearly polynomially-bounded in the size of the input representation of A.

For the base case, the algorithm $M_0^{a,b}$ first verifies that A meets the syntactic requirements of an η-simple PPS. If so, then we verify, given $\mathcal{T}_0 = (\tau_0, \sigma_0, j_0)$, that τ_0 has at most $a + b$ applications of the $Tick$ rule, is compliant, and leads to a goal. By Proposition 4, since τ_0 is polynomially-bounded in A, we can verify compliance of τ_0 in polynomial time in A. By Remark 4, we can verify in polynomial time in A, given (σ_0, j_0), that the last configuration of τ_0 is a goal. Hence $M_0^{a,b}(A, \mathcal{T}_0)$ runs in polynomial time, and A is $(0, a, b)$-resilient if and only if $\exists \mathcal{T}_0\ M_0^{a,b}(A, \mathcal{T}_0)$ accepts.

Now suppose inductively that we have, for each $a' \in \mathbb{Z}^+$ and $b' \in \mathbb{N}$, algorithms $M_k^{a',b'}$ satisfying (Eq. 6) with $n = k$. Fix some $a \in \mathbb{Z}^+$ and $b \in \mathbb{N}$, and we define an algorithm $M_{k+1}^{a,b}$ which takes inputs of the form $(A, \mathcal{T}, \mathcal{T}', \mathcal{T}_1, \ldots, \mathcal{T}_k, \rho, \rho_1, \ldots, \rho_k)$. Let $\mathcal{T} = (\tau, \sigma, j)$, $\mathcal{T}' = (\tau', \sigma', j')$, and $\rho = (r, \sigma^*, i)$. Furthermore, let t_0 denote the global time in the initial configuration $\mathcal{S}_0$, $|\tau|$ denote the length of τ, $\mathcal{S}'_{i+1}$ denote the initial configuration of τ', t_i denote the global time in the i^{th} configuration $\mathcal{S}_i$ of τ, and $d_i = t_i - t_0$. We now describe the run of $M_{k+1}^{a,b}$ on this input.

First, check that τ and τ' are compliant traces to a goal configuration. Then, check if $d_i \leq a$; if this check fails, then we halt and accept, since by Definition 11, update rules cannot be applied after more than a time steps. Then, check if $\mathcal{S}_i \longrightarrow_r \mathcal{S}'_{i+1}$, by applying the ground substitution σ^* to r and checking that it is applicable to $\mathcal{S}_i$. If this checks fails, then we halt and accept, since r is not an applicable update rule to $\mathcal{S}_i$. Otherwise, check that $\mathcal{S}'_{i+1}$ is the correct result of applying this instance of r to $\mathcal{S}_i$. If this check fails, then reject, since τ' cannot be a valid reaction trace. Finally, let $A' = (\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}'_{i+1})$, and simulate $M_k^{a-d_i,b}$ on the input $(A', \mathcal{T}', \mathcal{T}_1, \ldots, \mathcal{T}_k, \rho_1, \ldots, \rho_k)$. If the result of this simulation is that $M_k^{a-d_i,b}$ accepts the input, then we halt and accept, since by the inductive hypothesis, τ' must be a $(k, a - d_i, b)$-resilient reaction trace. Otherwise, we reject.

Taking into account the inductive hypothesis and Remark 4, it is clear that $M_{k+1}^{a,b}$ runs in polynomial time in the size of its input. Furthermore, it follows immediately by inspection of Definition 11 that A is $(k + 1, a, b)$-resilient if and only if

$$\exists \mathcal{T}_0 \forall \rho_1 \exists \mathcal{T}_1 \ldots \forall \rho_{k+1} \exists \mathcal{T}_{k+1}\ M_{k+1}^{a,b}(A, \mathcal{T}, \mathcal{T}_1, \ldots, \mathcal{T}_{k+1}, \rho, \rho_1, \ldots, \rho_{k+1}) \text{ accepts.}$$

This concludes the inductive argument. It follows immediately from the quantifier-alternation characterization of PH that the (n, a, b)-resilience problem for η-simple PPSs with traces containing only facts of bounded size is in $\Sigma_{2n+1}^{\mathsf{P}}$. □

Remark 5. Even without assuming η-simplicity, a slight variation of the above argument gives a decision procedure of complexity $\Sigma_{2n+2}^{\mathsf{P}}$ for the (n, a, b)-resilience problem for PPSs with traces containing facts of bounded size. To modify the argument, we allow each universal quantifier to range over an additional ground substitution, which is used in the verification algorithm $M_n^{a,b}$ to check that an arbitrary configuration in the preceding witness trace is non-critical. Note that this check can be done in polynomial time (cf. Remark 4). If this check succeeds for *all* configurations and all *all* such ground substitutions, then every witness trace is compliant.

In fact, even for 1-simple PPSs, the (n, a, b)-resilience problem is $\Sigma_{2n+1}^{\mathsf{P}}$-hard. We show this by a reduction from Σ_{2n+1}-SAT, the language of true quantified Boolean formulas (QBF) with $2n + 1$ quantifier alternations, where the first quantifier is existential

and the underlying propositional formula is in 3-CNF form. This problem is known to be $\Sigma^{\mathsf{P}}_{2n+1}$-complete [37]. Recall that the truth of a quantified Boolean formula can be analyzed by considering the *QBF evaluation game* for the formula. In this game, two players, *Spoiler* and *Duplicator*, take turns choosing assignments to the formula's quantified variables. Duplicator chooses assignments for existentially-quantified variables with the goal of satisfying the underlying Boolean formula, while Spoiler chooses assignments for universally-quantified variables with the goal of falsifying it. The game concludes once assignments have been chosen for all of the quantified variables. A QBF ψ is true if and only if Duplicator has a winning strategy in this game [34].

In our reduction, we encode the positions of this QBF evaluation game into configurations, where a position of the QBF evaluation game for a formula

$$\psi := \exists \overline{v}_1 \forall \overline{v}_2 \exists \overline{v}_3 \ldots \forall \overline{v}_{2n} \exists \overline{v}_{2n+1} \varphi(\overline{v}_1, \overline{v}_2, \overline{v}_3, \ldots, \overline{v}_{2n+1})$$

is a sequence $\mathcal{P} = V_1, \ldots, V_j$ of assignments to the variables in $\overline{v}_1, \ldots, \overline{v}_j$ for some $j \leq 2n+1$. If j is even, then we say that the position $\mathcal{P}$ *belongs to Duplicator*; otherwise, we say that it *belongs to Spoiler*. The player who owns a given position makes the next move, choosing an assignment for the variables in the tuple $\overline{v}_{j+1}$. We use system rules to model assignments made by Duplicator, while update rules are used to model assignments made by Spoiler. Intuitively, the goal configurations are those positions of the game which encode assignments satisfying the underlying formula φ.

Theorem 2. *For all $a \in \mathbb{Z}^+$ and $b \in \mathbb{N}$, there exists a polynomial-time reduction from the Σ_{2n+1}-SAT problem to the (n, a, b)-resilience problem. Furthermore, the computed instance is always a 1-simple progressing planning scenario with traces containing only facts of bounded size.*

Proof (sketch). Let $\psi := \exists \overline{v}_1 \forall \overline{v}_2 \exists \overline{v}_3 \ldots \forall \overline{v}_{2n} \exists \overline{v}_{2n+1} \varphi(\overline{v}_1, \overline{v}_2, \overline{v}_3, \ldots, \overline{v}_{2n+1})$ be an instance of Σ_{2n+1}-SAT, where the $\overline{v}_i$ are tuples of variables and φ is a 3-CNF formula. We can compute a 1-simple progressing planning scenario $A = (\mathcal{R}, \mathcal{GS}, \mathcal{CS}, \mathcal{E}, \mathcal{S}_0)$ which is (n, a, b)-resilient if and only if ψ is true. To do this, the initial configuration $\mathcal{S}_0$ contains 0-ary facts of the form Unk_i which indicate that the assignment to $\overline{v}_i$ is unknown, for each $1 \leq i \leq 2n+1$. We also include a 0-ary fact Rnd_0 indicating that no rounds of the QBF game have been played, and a 0-ary fact corresponding to each clause of φ. This represents the initial position of the QBF evaluation game.

We include system (resp. update) rules corresponding to assignments to the $\overline{v}_i$ tuples of variables for even (resp. odd); these rules consume the fact Unk_i and create facts of the form $Val_i(\overline{b})$, where $\overline{b}$ is a tuple Boolean values ($true$ or $false$). These rules simulate moves of the QBF evaluation game, and change the configuration to represent the next position of the game. Furthermore, each rule of this kind can only be played when the appropriate Rnd_i fact is in the current configuration, and it increments the round counter from Rnd_i to Rnd_{i+1}. This ensures that the players can only choose assignments from positions that belong to them. We also include "verification" rules, which are used to check if the assignment after the conclusion of the game (encoded by the Val_i facts) satisfies φ. The goal configurations are those in which the final assignment has been verified to satisfy φ.

This encoding does not depend on the parameters a and b of the resilience problem: A admits an $(n, 1, 0)$-resilient trace if and only if it admits an (n, a, b)-resilient trace for all $a \in \mathbb{Z}^+$ and $b \in \mathbb{N}$. It follows easily from the simulation of the QBF evaluation game that, for all $a \in \mathbb{Z}^+$ and $b \in \mathbb{N}$, A is (n, a, b)-resilient if and only if Duplicator has a winning strategy for the QBF evaluation game for the formula ψ. □

A detailed specification of the reduction can be found in the technical report [3]. Theorems 1 and 2 immediately entail the following.

Corollary 1. *The (n, a, b)-resilience problem for η-simple PPSs with traces containing only facts of bounded size is $\Sigma^{\mathsf{P}}_{2n+1}$-complete.*

5 Verifying Resilience in Maude

To experiment with resilience, we specified our running example of a travel planning scenario in the Maude rewriting logic language [11]. In contrast to the multiset rewriting representation, the Maude specification uses data structures, not facts, to represent system structure and state. The passing of time is modeled using rule duration. For example, the rule that models taking a flight takes time according to the duration of the flight. These rules combine an instantaneous rule with a time-passing rule. These design decisions, together with relegating as much as possible to equational reasoning, help reduce the search state space.

In the travel system scenarios, the goal is to attend a given set of events. System updates change flight schedules; goal updates either change event start time or duration, or add an event. Execution traces terminate when the last event is attended or an event is missed. Thus, for simplicity, we fix $a + b$ to be the end of the last possible event and leave it implicit. In the following, we describe the representation of key elements of the travel system specification: system state, execution rules, and updates. We then explain the algorithm for checking (n, a, b)-resilience and report on some simple experiments.

Representing travel status. The two main sorts in the travel system specification are `Flight` and `Event`. A term `fl(cityD,cityA,fn,depT,dur)` represents a flight, where `cityD` and `cityA` are the departure and arrival cities, `fn` is the flight number (a unique identifier), `depT` is the departure time, and `dur` is the duration. The departure time and duration are represented by hour-minute terms `hm(h,m)`. For simplicity, flights are assumed to go at the same time every day, and all times are in GMT. A flight instance (sort `FltInst`) represents a flight on a specific date by a term: `fi(flt,dtDep,dtArr)` where `flt` is a flight, `dtDep` and `dtArr` are date-time terms representing the date and time of departure and arrival respectively. A date-time term has the form `dt(yd,hm)` where `yd` is a year-day term `yd(y,d)` and `hm` is an hour-minute term as above.

An event is represented by a term `ev(eid,city,loc,yd,hm,dur,opt)`, where `eid` is a unique (string) identifier and the `opt` Boolean specifies if the event is optional; the other arguments are as above. A term of the form `{conf}` (sort `Sys`), where `conf` (sort `Conf`) is a multiset of configuration elements, represents a system execution state.

The sorts of configuration elements are `TConf`, `Log`, and update descriptions. Terms of sort `TConf` represent a traveler's state, with one of three forms:

`tc(dt,city,loc,evs)` – planning
`tc(dt,city,loc,evs,ev,fltil)` – executing
`tcCrit(dt,city,loc,evs,ev,reason)` – critical

Here, `dt` is a date-time term, the travelers current date and time, and `city` tells what city the traveler is currently in. The term `loc` gives the location within the city, either the `airport` or the city `center`. The term `evs` is the set of events to be attended, while `ev` is the next event to consider. The term `fltil` is the flight instance list chosen to get from `city` to the location of `ev`. The constructor `tcCrit` signals a critical configuration in which a required event has been missed.

An update description is a term of the form `di(digs)` or `di(digs,n)` where `digs` is a set of digressions. Each digression describes an update to be applied by an update rule, and `n` bounds the number of updates that can be applied. Two kinds of updates are currently implemented: flight/system updates and event/goal updates. The flight updates are: `cancel`, which cancels the current flight, `delay(hm(h,m))`, which delays the current flight by `h` hours, `m` minutes, and `divert(city0,city1)`, which diverts the current flight from `city0` to `city1`, where the current flight is the first element of the flight instance list of an executing `TConf` term. The event/goal updates are: `edEvS(hm(h,m)`, which starts the current event `h` hours, `m` minutes earlier, `edEvD(hm(h,m))`, which extends the current event duration by `h` hours, `m` minutes, and `addEv(eid)`, which adds the event with id `evid` to the set of pending events.

Flight updates are only applied to the next flight the traveler is about to take. Similarly, the changes in event start time or duration are only applied to the next event to attend. This is a simplified setting, but sufficiently illustrates our formalism; more complex variations are possible. Lastly, an element of sort `Log` is a list of log items. It is used to record updates, flights taken, and events attended or missed. Among other things, when searching for flights, it is used to know which flights have been canceled.

Rewrite rules. There are five system rules (`plan`, `noUFlts`, `flt`, `event`, and `replan`) and two update rules (`fltDigress` and `evDigress`, for flight/system updates and event/goal updates, respectively). The `plan` rule picks the event, `ev`, with the earliest start time from `nevs` and (non-deterministically) selects a list of flight instances, `fltil`, from the set of flight instance lists arriving at the event city before the start time. The set of possible flights is stored in a constant `FltDB`. `log1` is `log` with an item recording the rule firing added. The conditional `if ...` does the above computing.

```
crl [plan]: {tc(dt,city, loc, nevs) log}
          => {tc(dt, city, loc, evs0, ev, fltil) log1} if ...
```

The rule `noUFlts` handles cases in which there is no usable flight instance list given the traveler time and location and the time and location of the next event. If `ev` is optional then it is dropped (recording this in the log) and the rule `plan` is applied to the remaining events; otherwise, the configuration becomes critical and execution terminates. The rule `flt` models taking the next flight, assuming the flight departure time is later than the traveler's current time. This rule updates the traveler's time to the flight arrival time `dtArr` and traveler's city to the destination `city1`. Then, the flight

instance taken is removed from the list.

```
crl [flt]: {tc(dt, city0, airport, evs,ev,flti ; fltil) conf}
  => {tc(dtArr, city1, airport, evs,ev, fltil) conf1} if ....
```

The `event` rule (not shown) models attending the currently selected event. It can be applied when the traveler city is the same as the event city and the current time is not after the event start. As for the flight rule, the current time is updated to the event end time and the traveler returns to the airport. If the traveler arrives at the event city too late, as for the `noUFlts` rule, if the event is optional, the `event` rule drops the event and enters a log item, otherwise it produces a critical `TConf`. The `replan` rule handles the situation in which the traveler city is not the event city and the next flight, if any, does not depart from the traveler city or has been missed. The pending flight instance list is dropped, the event is put back in the event set, and a log item is added to the log.

The update rule `fltDigress` only applies if the digression counter is greater than zero and the rule decrements the counter. A flight digression, `fdig`, is non-deterministically selected from the available digressions (the first argument to `di`).

```
crl [fltDigress]: {tconf di(fdig digs, s n) conf}
=> {tconf1 di(digs fdig,n) conf1}
   if tc(dt, city0, airport, evs,ev, flti ; fltil) := tconf
   /\ city0 =/= getCity(ev)
   /\ tconf1 conf1 := applyDigression(tconf,conf,fdig)
```

The first condition exposes the structure of `tconf` to ensure there is a pending flight to update. The auxiliary function `applyDigression` specifies the result of the update. For example, the `cancel` update removes `flti` from the list and adds a log item recording that this flight instance is cancelled. The case where the update description has the form `di(fdig digs)` is similar, except here `fdig` is removed when applied, and updating stops when there are no more update elements in the set. Similarly, the rule `evDigress` non-deterministically selects an event digression from the configuration's digression set and applies the auxiliary function `applyEvDigress` to determine the effect of the update. It only applies if the update counter is greater than zero, and the `TConf` component has a selected next event.

A planning scenario is defined by an initial system configuration `iSys`, a database of flights `FltDB`, a database of events `EvDB`, and an update description. A trace `iSys -TR-> xSys` is a sequence of applications of rule instances from the travel rules `TR`, leading from `iSys` to `xSys`. It is a *compliant goal trace* if `xSys` satisfies the goal condition (`goal`) that the traveler configuration has no remaining events, and no required events have been missed. Formally, the traveler component of `xSys` must have the form `tc(dt,city,loc,mtE)`, since, when a required event is missed, it is rewritten to a term of the form `tcCrit(dt,city,loc,evs,ev,comment)`.

Checking (n, a, b)-resilience. Checking (n, a, b)-resilience in the travel planning system is implemented by the equationally-defined function `isAbRes` using Maude's reflection capability and strategy language. As previously mentioned, the upper bound on time is implicitly determined by the times and durations of available events, not treated as a parameter.

N:	1		2		3	
2ev	**R?**	**time**	**R?**	**time**	**R?**	**time**
247	N	86ms	-	-	-	-
246	Y	81ms	Y	147ms	N	7476ms
3ev	**R?**	**time**	**R?**	**time**	**R?**	**time**
247	N	1400ms	-	-	-	-
246	Y	325ms	Y	685ms	NF	-

(a) flight/system update rules

N:	1		2		3	
2ev	**R?**	**time**	**R?**	**time**	**R?**	**time**
247	Y	78ms	N	77ms	-	-
246	Y	98ms	N	34800ms	-	-
3ev	**R?**	**time**	**R?**	**time**	**R?**	**time**
247	Y	143ms	N	2627ms	-	-
246	Y	220ms	Y	633ms	Y	2634ms

(b) event/goal update rules

Fig. 2: Summary of (n, a, b)-resilience experiments

The function `isAbRes` checks (n, a, b)-resilience by first using `metaSearch` to find a candidate goal state, then `metaSearchPath` gives the corresponding compliant goal trace[6] The candidate trace is converted to a rewrite strategy (representing the trace's list of rule instances). The function `checkAbRes` iterates through the initial prefixes of the strategy, using `metaSRewrite` to follow the trace prefix. This implements a check for reaction traces at all possible points of update rule application (cf. Definition 11). For each state resulting from executing a prefix, the function `checkDigs` is called to apply each one of the available updates, and then we invoke `isAbRes` to check for an $(n-1, a-d, b)$-resilient extension trace, where d is the number of time steps up to the end of the prefix. If n is zero, `abResCheck` simply finds a compliant goal trace. If no $(n-1, a-d, b)$-resilient extension trace can be found, then the current candidate trace is rejected, and `isAbRes` continues searching for the next candidate trace. If the search for candidate traces fails, then the system under consideration is not (n, a, b)-resilient. If the check for an $(n-1, a-d, b)$-resilient extension trace succeeds for every update of every prefix execution, then the strategy is returned as a witness for (n, a, b)-resilience. We tested (n, a, b)-resilience to flight/system updates and event/goal updates with instances of the following command.

```
red isAbRes(['TRAVEL-SCENARIO],N,allDi,SYST,patT,tCond,uStrat,0).
red isAbRes(['TRAVEL-SCENARIO],1,allEv,iSysT,patT,tCond,ueStrat,0).
```

The results are summarized in Table 2. Note that `N` is the number of updates (1, 2, or 3), `SYST` is (the meta representation of) an initial state with a starting day that is as late as possible to succeed if nothing goes wrong (247) or one day earlier (246) and 2 or 3 events. `patT` and `tCond` are the metaSearch pattern and condition arguments and `uStrat` is used to construct the update rule strategy. In the summary tables the `R?` indicates the result of `isAbRes`: `Y` for yes (a non-empty strategy is returned) and `N` for no. A dash indicates the experiment not done (because the check fails for smaller N). Lastly, `allEv` is the set of all implemented event update descriptions.

6 Conclusions and Related Work

We have shown that, for η-simple PPSs with traces containing only facts of bounded size, the (n, a, b)-resilience problem is $\Sigma^{\mathsf{P}}_{2n+1}$-complete. In [1], we showed that the

[6] Execution terminates if a critical state is reached, so paths to goal states are always compliant.

version of this problem without time bounds is PSPACE-complete for balanced systems with traces containing only facts of bounded size. In addition to the formal model and complexity results, we have implemented automated verification of time-bounded resilience using Maude. Resilience has been studied in diverse areas such as civil engineering [8], disaster studies [29], and environmental science [16]. Formalizations of resilience are often tailored to specific applications [36,4,30,19,27] and cannot be easily adapted to different systems. However, while we illustrated time-bounded resilience via an example of a PPS modeling a flight planning scenario, our earlier work has studied similar properties for a diverse range of other critical, time-sensitive systems, from collaborative systems subject to governmental regulations [24], to distributed unmanned aerial vehicles (UAV) performing safety-critical tasks [21,22]. The strength of our formalism is its flexibility in modeling a wide range of multi-agent systems.

Interest in resilience, as well as other related concepts such as robustness [9], recoverability [22,32], fault tolerance [25], and reliability [5], has grown in recent years. In [32], the authors introduced a notion of *time-bounded recovery* for *logical scenarios*, which are families of system states represented by patterns whose variables are constrained to describe an operating domain, and where recoverability is parameterized by an ordered set of safety conditions. Intuitively, a t-recoverable logical scenario is one which, when operating in normal mode, can recover from a lower-level safety condition to an optimal safety condition within t time steps, without reaching an unsafe condition. Like our formalization of resilience, t-recoverability concerns recovery from a deviation from normal execution. The primary distinction in [32] is that deviations are internal to the system, rules update the model state and compute control commands, and enabled rules must fire before time passes, as is common in real-time systems.

Our definition of time-bounded resilience (Definition 11) can be seen as a modification of [1, Definitions 9-10], with time parameters a and b and taking into account both system and goal updates. Here, the recoverability conditions from [1] are simplified to the total relation on configurations. Further investigation of recoverability conditions and resilience with respect to update rules that consume or create critical facts is left for future work. Another avenue of future work is to find conditions beyond η-simplicity which allow for polynomial-time solvability of the trace compliance problem. We also plan to study time-bounded resilience problems with respect to update rules that consume or create critical facts, and other variations involving, *e.g.*, real-time models and infinite traces. We are interested in the relationship between resilience and other properties of time-sensitive distributed systems [22], such realizability, recoverability, reliability, and survivability, as well as in specific applications of resilience, where further implementation results could provide interesting insights.

Acknowledgments. We thank Vivek Nigam for many insightful discussions during the early part of this work. Kanovich was partially supported by EPSRC Programme Grant EP/R006865/1: "Interface Reasoning for Interacting Systems (IRIS)". Scedrov was partially supported by the U. S. Office of Naval Research under award number N00014-20-1-2635 during the early part of this work. Talcott was partially supported by the U. S. Office of Naval Research under award numbers N00014-15-1-2202 and N00014-20-1-2644, and NRL grant N0017317-1-G002.

References

1. M. A. Alturki, T. Ban Kirigin, M. Kanovich, V. Nigam, A. Scedrov, and C. Talcott. On the formalization and computational complexity of resilience problems for cyber-physical systems. In *Theoretical Aspects of Computing–ICTAC 2022: 19th International Colloquium, Tbilisi, Georgia, September 27–29, 2022, Proceedings*, pages 96–113. Springer, 2022.
2. S. Arora and B. Barak. *Complexity theory: A modern approach*. Cambridge University Press Cambridge, 2009.
3. T. Ban Kirigin, J. Comer, M. Kanovich, A. Scedrov, and C. Talcott. Technical report: Time-bounded resilience. *arXiv preprint arXiv:2401.05585*, 2024.
4. S. Banescu, M. Ochoa, and A. Pretschner. A framework for measuring software obfuscation resilience against automated attacks. In *2015 IEEE/ACM 1st International Workshop on Software Protection*, pages 45–51, 2015.
5. E. Bauer. *Design for reliability: information and computer-based systems*. John Wiley & Sons, 2011.
6. A. Bennaceur, C. Ghezzi, K. Tei, T. Kehrer, D. Weyns, R. Calinescu, S. Dustdar, Z. Hu, S. Honiden, F. Ishikawa, Z. Jin, J. Kramer, M. Litoiu, M. Loreti, G. Moreno, H. Müller, L. Nenzi, B. Nuseibeh, L. Pasquale, W. Reisig, H. Schmidt, C. Tsigkanos, and H. Zhao. Modelling and analysing resilient cyber-physical systems. In *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 70–76, 2019.
7. R. Bloomfield, G. Fletcher, H. Khlaaf, P. Ryan, S. Kinoshita, Y. Kinoshit, M. Takeyama, Y. Matsubara, P. Popov, K. Imai, et al. Towards identifying and closing gaps in assurance of autonomous road vehicles–a collection of technical notes part 1. *arXiv preprint arXiv:2003.00789*, 2020.
8. A. Bozza, D. Asprone, and F. Fabbrocino. Urban resilience: A civil engineering perspective. *Sustainability*, 9(1), 2017.
9. M. Bruneau, S. E. Chang, R. T. Eguchi, G. C. Lee, T. D. O'Rourke, A. M. Reinhorn, M. Shinozuka, K. Tierney, W. A. Wallace, and D. Von Winterfeldt. A framework to quantitatively assess and enhance the seismic resilience of communities. *Earthquake spectra*, 19(4):733–752, 2003.
10. S. Caminiti, I. Finocchi, E. G. Fusco, and F. Silvestri. Resilient dynamic programming. *Algorithmica*, 77(2):389–425, Feb 2017.
11. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *All About Maude: A High-Performance Logical Framework*, volume 4350 of *LNCS*. Springer, 2007.
12. D. Cunningham, D. Grove, B. Herta, A. Iyengar, K. Kawachiya, H. Murata, V. Saraswat, M. Takeuchi, and O. Tardieu. Resilient x10: Efficient failure-aware programming. *SIGPLAN Not.*, 49(8):67–80, feb 2014.
13. N. A. Durgin, P. Lincoln, J. C. Mitchell, and A. Scedrov. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, 12(2):247–311, 2004.
14. O. Eigner, S. Eresheim, P. Kieseberg, L. D. Klausner, M. Pirker, T. Priebe, S. Tjoa, F. Marulli, and F. Mercaldo. Towards resilient artificial intelligence: Survey and research issues. In *2021 IEEE International Conference on Cyber Security and Resilience (CSR)*, pages 536–542, 2021.
15. U. Ferraro-Petrillo, I. Finocchi, and G. F. Italiano. Experimental study of resilient algorithms and data structures. In P. Festa, editor, *Experimental Algorithms*, pages 1–12, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
16. C. Folke. Resilience: The emergence of a perspective for social–ecological systems analyses. *Global Environmental Change*, 16(3):253–267, 2006. Resilience, Vulnerability, and

Adaptation: A Cross-Cutting Theme of the International Human Dimensions Programme on Global Environmental Change.
17. S. Goel, S. Hanneke, S. Moran, and A. Shetty. Adversarial resilience in sequential prediction via abstention. *arXiv preprint arXiv:2306.13119*, 2023.
18. Y. Hirshfeld and A. Rabinovich. Logics for real time: Decidability and complexity. *Fundamenta Informaticae*, 62(1):1–28, 2004.
19. W. Huang, Y. Zhou, Y. Sun, A. Banks, J. Meng, J. Sharp, S. Maskell, and X. Huang. Formal verification of robustness and resilience of learning-enabled state estimation systems for robotics, 2020.
20. S. Hukerikar, P. C. Diniz, and R. F. Lucas. A programming model for resilience in extreme scale computing. In *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)*, pages 1–6, 2012.
21. M. Kanovich, T. Ban Kirigin, V. Nigam, A. Scedrov, and C. Talcott. Timed multiset rewriting and the verification of time-sensitive distributed systems. In *14th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, 2016.
22. M. Kanovich, T. Ban Kirigin, V. Nigam, A. Scedrov, and C. Talcott. On the complexity of verification of time-sensitive distributed systems. In D. Dougherty, J. Meseguer, S. A. Mödersheim, and P. Rowe, editors, *Protocols, Strands, and Logic*, volume 13066 of *Springer LNCS*, pages 251–275. Springer International Publishing, 2021.
23. M. Kanovich, T. Ban Kirigin, V. Nigam, A. Scedrov, and C. L. Talcott. Time, computational complexity, and probability in the analysis of distance-bounding protocols. *Journal of Computer Security*, 25(6):585–630, 2017.
24. M. Kanovich, T. Ban Kirigin, V. Nigam, A. Scedrov, C. L. Talcott, and R. Perovic. A rewriting framework and logic for activities subject to regulations. *Mathematical Structures in Computer Science*, 27(3):332–375, 2017.
25. I. Koren and C. M. Krishna. *Fault-tolerant systems*. Morgan Kaufmann, 2020.
26. X. Koutsoukos, G. Karsai, A. Laszka, H. Neema, B. Potteiger, P. Volgyesi, Y. Vorobeychik, and J. Sztipanovits. Sure: A modeling and simulation integration platform for evaluation of secure and resilient cyber–physical systems. *Proceedings of the IEEE*, 106(1):93–112, 2018.
27. A. M. Madni, D. Erwin, and M. Sievers. Constructing models for systems resilience: Challenges, concepts, and formal methods. *Systems*, 8(1), 2020.
28. A. M. Madni and S. Jackson. Towards a conceptual framework for resilience engineering. *IEEE Systems Journal*, 3(2):181–191, 2009.
29. S. B. Manyena. The concept of resilience revisited. *Disasters*, 30(4):434–450, 2006.
30. S. Mouelhi, M.-E. Laarouchi, D. Cancila, and H. Chaouchi. Predictive formal analysis of resilience in cyber-physical systems. *IEEE Access*, 7:33741–33758, 2019.
31. R. Neches and A. M. Madni. Towards affordably adaptable and effective systems. *Systems Engineering*, 16(2):224–234, 2013.
32. V. Nigam and C. L. Talcott. Automating recoverability proofs for cyber-physical systems with runtime assurance architectures. In C. David and M. Sun, editors, *17th International Symposium on Theoretical Aspects of Software Engineering*, volume 13931 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2023.
33. F. O. Olowononi, D. B. Rawat, and C. Liu. Resilient machine learning for networked cyber physical systems: A survey for machine learning security to securing machine learning for cps. *IEEE Communications Surveys & Tutorials*, 23(1):524–552, 2021.
34. C. H. Papadimitriou. *Computational complexity*. Academic Internet Publ., 2007.
35. A. Prasad. *Towards Robust and Resilient Machine Learning*. PhD thesis, Carnegie Mellon University, Apr 2022.
36. V. C. Sharma, A. Haran, Z. Rakamaric, and G. Gopalakrishnan. Towards formal approaches to system resilience. In *2013 IEEE 19th Pacific Rim International Symposium on Dependable Computing*, pages 41–50, 2013.

37. L. J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.
38. M. Vardi. Efficiency vs. resilience: What covid-19 teaches computing. *Communications of the ACM*, 63(5):9–9, 2020.