Access Controlled Temporal Networks

Carlo Combi[1], Roberto Posenato[1], Luca Viganò[2] and Matteo Zavatteri[1]
[1]*Department of Computer Science, University of Verona, Verona, Italy*
[2]*Department of Informatics, King's College London, London, U.K.*

Keywords: Access Control, Temporal Networks, Temporal Authorization, Dynamic Controllability, UPPAAL-TIGA.

Abstract: We define Access-Controlled Temporal Networks (ACTNs) as an extension of Conditional Simple Temporal Networks with Uncertainty (CSTNUs). CSTNUs are able to handle features such as contingent durations and conditional constraints, and have thus been used to model the temporal constraints of workflows underlying business processes. However, CSTNUs are unable to model users and authorization constraints, and thus cannot model "who can do what, when". ACTNs solve this problem by adding users and authorization constraints that must be considered together with temporal constraints. Dynamic controllability (DC) of ACTNs ensures the existence of an execution strategy, able to assign tasks to authorized users dynamically, satisfying all the relevant authorization constraints no matter what contingent durations turn out to be or what conditional constraints have to be considered. We show that the DC checking can be done via Timed Game Automata and provide experimental results using UPPAAL-TIGA on a concrete real-world case study.

1 INTRODUCTION

Context and Motivation. Workflow technology has emerged as one of the leading technologies for modeling, (re)designing, and executing business processes in several different application domains. For instance, workflows have been used to model processes for industrial R&D, manufacturing, energy distribution, banking processes, critical infrastructures and healthcare (Lenz and Reichert, 2007; Combi *et al.*, 2014a).

The formal modeling of workflows has been investigated in quite some detail and, depending on the characteristics the designers want to focus on, different formal models have been employed to verify interesting properties of workflow instances. Recent research has in particular investigated the modeling of temporal workflows by means of *Conditional Simple Temporal Networks with Uncertainty* (*CSTNUs*, (Combi *et al.*, 2014a; Hunsberger *et al.*, 2012)), which can handle temporal plans subject to both conditional constraints and contingencies simultaneously.

Time points (modeling the occurrence of the events upon their "execution") and constraints between pairs of time points are *labeled* by a conjunction of propositions saying when they must be considered. Each proposition is associated to an *observation time point*, a special type of time point that assigns either true or false to the proposition upon its execution.

Instead, a contingent duration between two time points A and C represents a range (i.e., a real interval) of allowed durations between A and C that cannot be fixed by the executing agent (i.e., the controller). A is the *activation time point* and it is under the control of this agent, whereas C is the *contingent time point*, and it is not. That is, whatever the duration turns out to be, the time instant in which the uncontrollable event modeled by the contingent link ends (i.e., C executes) is only observed in real time.

CSTNUs can handle temporal plans subject to both conditional constraints and contingencies. In a CSTNU, the truth values of propositions and the contingent durations are observed in real time as the network executes. That is, before executing the network, we do not know what constraints and time points will be considered nor how long the contingent durations will last. Hence, to deal with both contingent durations and truth value assignments, which are *uncontrollable*, the assignment of a task to a user must be done in real time as different behaviors of such uncontrollable parts might (and most of the time do) entail different assignments. If we pre-computed such an assignment before starting, we would risk violating some constraint. The analysis must thus be boiled down to *controllability* and not to mere consistency in which scheduling tasks and deciding which users will carry them out before the start of the execution would be sufficient.

For these reasons, the "temporal networks" com-

Combi C., Posenato R., ViganÃš L. and Zavatteri M.
Access Controlled Temporal Networks.
DOI: 10.5220/0006185701180131
In *Proceedings of the 9th International Conference on Agents and Artificial Intelligence (ICAART 2017)*, pages 118-131
ISBN: 978-989-758-220-2

munity has devoted considerable interest to the analysis of *dynamic controllability* (DC): a CSTNU is dynamically controllable if there exists an execution strategy that executes all the parts under control of the network by making decisions in real time depending on how the uncontrollable parts behave. The aim of this strategy is that of eventually satisfying all relevant constraints. DC analysis is done by either dedicated network-based algorithms (Combi *et al.*, 2013; Combi *et al.*, 2014b) or translation to *Timed Game Automata (TGAs)* (Cimatti *et al.*, 2016).

However, CSTNUs fail to model access-controlled workflows as they do not deal with users and authorization constraints. To address this problem, CSTNUs need to be extended by first injecting users and authorization constraints and then adapting the DC checking to answer to the *workflow satisfiability problem (WSP)*. WSP is the problem of finding an assignment of tasks to users so that the execution gets to the end without violating any authorization constraint (Wang and Li, 2010). In a temporal context we must also say when such an assignment takes place.

Such an extension allows one to deal with situations in which users, authorization constraints and temporal constraints *must* be considered all together and not as disjoint at all. In fact, workflows and access control have traditionally been approached as orthogonal, independent formalisms, but there are a large number of relevant cases in which they cannot be considered so. For example, a surgeon, who has just carried out a 4-hour intervention, is allowed to do another one only after resting from 2 to 4 hours. Likewise, a truck driver must rest for at least 45 minutes after a driving shift of 4.5 hours. These temporal constraints must be considered *in conjunction* with access control as there is a mutual influence.

Contributions. Our contributions are three-fold. First, we define *Access-Controlled Temporal Networks (ACTNs)* as an extension of CSTNUs by the injection of users and authorization constraints, motivated by the need of handling workflows subject to both temporal constraints and access control simultaneously. Each authorization constraint expresses a temporal range, a label modeling the conditional part, and an authorization policy expressing which the authorized users are.

Second, we extend the encoding for CSTNUs into TGAs and use sound and complete reachability algorithms provided for TGAs by the UPPAAL-TIGA software (Behrmann *et al.*, 2007) to check whether an ACTN is DC. The encoding we define is *fully automated* and generates the TGA in polynomial time.

Third, we discuss our approach by means of a concrete, real-world case study from the e-health domain implemented in UPPAAL-TIGA.

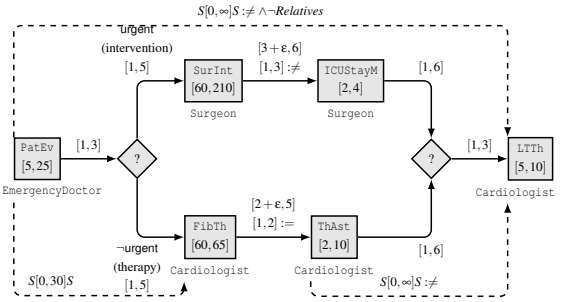

Figure 1: Official STEMI guidelines (excerpt). We model TSoD as $[1,3] :\neq$, TBoD as $[1,2] :=$. Ranges $[x,y]$ and $[x+\varepsilon,y]$ model all durations t s.t. $x \leq t \leq y$ and $x < t \leq y$, respectively.

Organization. § 2 introduces our case study. § 3 reviews essential background on CSTNUs and the encoding into TGAs. § 4 defines ACTNs along with their execution semantics. § 5 extends the encoding into TGAs given for CSTNUs. § 6 discusses related work. § 7 sums up and discusses future work.

2 THE STEMI GUIDELINES

As a simple but real-world (thus fully realistic) case study, we consider a simplification of the official guidelines for the treatment of STEMI patients (i.e., patients with ST-Elevation Myocardial Infarction), published by the American College of Cardiology/American Heart Association (see the discussion in (Combi *et al.*, 2014a)). As we discuss below, none of the previous works can handle such a "simple" example as they are not expressive (or formal) enough.

Fig. 1 shows the workflow; task durations, delays and temporal ranges are in minutes. The workflow starts with a patient evaluation (`PatEv`), in which an `EmergencyDoctor` establishes if the patient is in need of immediate medical attention. If the patient is urgent, a surgery intervention (`SurInt`) takes place. As soon as the intervention has finished, a `Surgeon` has to manage the patient's stay in the Intensive Care Unit (`ICUStayM`). This task lasts at least 2 minutes and at most 4. In addition, following the *Temporal Separation of Duties (TSoD)* policy, whenever `ICUStayM` starts within 3 minutes since the end of `SurInt`, the `Surgeon`s executing the two tasks must be different. TSoD does not apply if this temporal distance is greater than 3.

If the patient is not urgent, a fibrinolytic therapy (`FibTh`) takes place. This task lasts at least 60 and at most 65 minutes, and according to the guide-

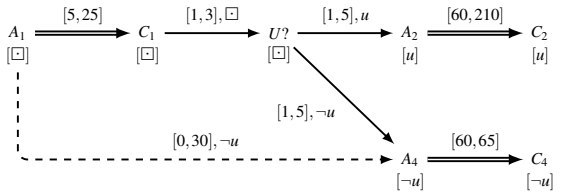

Figure 2: CSTNU modeling a portion of the workflow given in Fig. 1. $(A_1,5,25,C_1)$ models PatEv, $(A_2,60,210,C_2)$ models SurInt and $(A_4,60,65,C_4)$ models FibTh. $U?$ models the conditional split.

lines, it has to start within 30 minutes since PatEv has started. After FibTh has terminated, a therapy assessment (ThAst) starts with minimal and maximal allowed durations 2 and 10 minutes, respectively. Following the *Temporal Binding Of Duties (TBoD)* policy, if the start of ThAst is within 2 minutes since the end of FibTh, then the same Cardiologist must execute the two tasks. TBoD does not apply if this temporal distance is greater than 2 minutes.

Regardless of the chosen branch, after the last task in that branch has finished, a long term therapy assessment (LTTh) starts. The minimal and maximal durations for this task are 5 and 10 minutes, respectively. A security policy requires that the Cardiologist executing LTTh must be (i) different from and not a relative of the one who executed PatEv, and different from the one who executed ThAst if the patient was not urgent.

We will consider a *role-based access control environment* in which EmergencyDoctor contains users John and Lara, Surgeon contains Bob, Tom and Sara, and Cardiologist contains Lara, Kate and Rick.

3 CSTNUs

In this section, we provide background on CSTNUs that will be essential to understand our ACTNs.

Given a set $\mathcal{P}$ of propositional letters, a *label* ℓ is any (possibly empty) conjunction of literals, where a literal is either a propositional letter $p \in \mathcal{P}$ or its negation $\neg p$. The *empty label* is denoted by $\boxdot$. The *label universe of* $\mathcal{P}$, denoted by $\mathcal{P}^*$, is the set of all possible labels representing all possible (finite) conjunctions of literals we can obtain from $\mathcal{P}$ e.g., the label universe of $\mathcal{P} = \{p,q\}$ is $\mathcal{P}^* = \{\boxdot, p, q, \neg p, \neg q, p \wedge q, p \wedge \neg q, \neg p \wedge q, \neg p \wedge \neg q\}$. Two labels $\ell_1, \ell_2 \in \mathcal{P}^*$ are *consistent* if and only if their conjunction $\ell_1 \wedge \ell_2$ is satisfiable and a label ℓ_1 *entails* a label ℓ_2 (written $\ell_1 \Rightarrow \ell_2$) iff all literals in ℓ_2 appear in ℓ_1 too (i.e., if ℓ_1 is more *specific* than ℓ_2). Any label $\ell \in \mathcal{P}^*$ entails both itself (i.e., $\ell \Rightarrow \ell$) and the empty label (i.e., $\ell \Rightarrow \boxdot$).

Definition 1. A *Conditional Simple Temporal Network with Uncertainty* (*CSTNU*, (Hunsberger *et al.*, 2012)) is a tuple $\langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, O, \mathcal{P}, \mathcal{CT}, \mathcal{L} \rangle$, where:

(1) $\mathcal{T}$ is a finite set of *time points*.
(2) $\mathcal{P}$ is a finite set of propositional letters.
(3) $L\colon \mathcal{T} \to \mathcal{P}^*$ is a function assigning a label to each time point in $\mathcal{T}$ saying in what scenarios the time point must be considered.
(4) $\mathcal{OT} \subseteq \mathcal{T}$ is a set of *observation time points*, where each $P? \in \mathcal{OT}$ is associated to a proposition p, whose truth value is *unknown* before $P?$ executes.
(5) $O\colon \mathcal{P} \to \mathcal{OT}$ is a bijection associating a unique observation time point to each propositional letter.
(6) $\mathcal{CT} \subset \mathcal{T}$ is a set of contingent time points such that $\mathcal{OT} \cap \mathcal{CT} = \emptyset$.
(7) $\mathcal{L}$ is a set of contingent links each one having the form (A,x,y,C), where $A \in \mathcal{T}$ and $C \in \mathcal{CT}$ are different time points (written $A \not\equiv C$), $x,y \in \mathbb{R}$ $(0 < x < y < \infty)$, and $L(A) = L(C)$. Once A is executed, C is only observed to occur (as it is out of control). However, C is guaranteed to occur such that $C - A \in [x,y]$. For any pair (A_1,x_1,y_1,C_1), $(A_2,x_2,y_2,C_2) \in \mathcal{L}$, $C_1 \not\equiv C_2$.
(8) $\mathcal{C}$ is a set of *labeled constraints* each one having the form $(Y - X \leq k, \ell)$, where $X,Y \in \mathcal{T}$, $k \in \mathbb{R}$ and $\ell \in \mathcal{P}^*$. Many labeled constraints between two time points X and Y can be specified, provided that their labels are different.
(9) $\ell \Rightarrow L(Y) \wedge L(X)$ for each $(Y - X \leq k, \ell) \in \mathcal{C}$ (*constraint label coherence*), and $\ell \Rightarrow L(O(p))$ for each p or $\neg p$ appearing in ℓ (*constraint label honesty*) (Hunsberger *et al.*, 2015).
(10) For each $X \in \mathcal{T}$, if p or $\neg p$ appears in X's label, then $L(X) \Rightarrow L(O(p))$, and $(O(p) - X \leq -\varepsilon, L(X)) \in \mathcal{C}$ for some $\varepsilon > 0$ (*time point label honesty*) (Hunsberger *et al.*, 2015).

We represent a CSTNU as a directed multi-graph where the set of nodes coincides with the set of time points, whereas the set of edges is partitioned in requirement and contingent links. We draw a *requirement link*[1] as $X \xrightarrow{[x,y],\ell} Y$ meaning that once X executes, Y must execute such that $Y - X \in [x,y]$ if ℓ holds. We draw a *contingent link* $(A,x,y,C) \in \mathcal{L}$ as $A \overset{[x,y]}{\Longrightarrow} C$, where the label $[x,y]$ implicitly means $[x,y],\ell$ with $\ell = L(A) = L(C)$.

Fig. 2 shows an example of a CSTNU modeling a portion of the workflow in Fig. 1 without access control. We mapped PatEv into $(A_1,5,25,C_1)$, SurInt into $(A_2,60,210,C_2)$, FibTh into $(A_4,60,65,C_4)$, and the conditional split connector (diamond) into the observation time point $U?$. Once executed, $U?$ sets

[1] $X \xrightarrow{[x,y],\ell} Y$ models $(Y - X \leq y, \ell), (X - Y \leq -x, \ell) \in \mathcal{C}$.

the value of the proposition u and, thus, some constraints could be disabled (those having label inconsistent with u). We point out it is possible to decide when to execute $U?$ but not the truth value for u.

DC has been recently investigated by using TGAs, allowing one to work with *sound* and *complete* algorithms, and to synthesize memoryless execution strategies (Cimatti *et al.*, 2016). In particular, DC is modeled as a game between two players: the controller `ctrl` executing all non-contingent time points and the environment `env` executing the contingent ones and assigning truth values to propositions. The purpose of `ctrl` is to get to the `goal` location reachable as soon as all time points have been executed and all constraints have been satisfied with respect to the specific scenario, whereas that of the `env` is to prevent `ctrl` from getting there. If `env` wins, the network is *not* DC, otherwise it is.

Definition 2. A *Timed Automaton (TA)* consists of a finite set of *locations Loc* (with l_0 the initial one), a finite set of *actions Act*, a finite set of *real-value clocks* $\mathcal{X}$, a transition relation $\rightarrow \subseteq Loc \times \mathcal{H}(\mathcal{X}) \times Act \times 2^{\mathcal{X}} \times Loc$, where elements in $\mathcal{H}(\mathcal{X})$ are conjunctions of clock constraints of the form $x \bowtie k$ or $y - x \bowtie k$ with $x, y \in \mathcal{X}$, $k \in \mathbb{Z}$ and $\bowtie \in \{<, \leq, =, >, \geq\}$, and a function $Inv : Loc \rightarrow \mathcal{H}(\mathcal{X})$ assigning an *invariant* (i.e., a conjunction of clock constraints) to each location.

A *Timed Game Automaton (TGA)* extends a TA by partitioning the set of transitions in *controllable* and *uncontrollable* transitions.

In any run, all clocks start at zero and evolve uniformly. Each *transition* has the form $(l_i; G; N; R; l_j)$, where l_i is the start location, G is a guard made of a conjunction of atomic constraints $x \bowtie y$ over clocks, N is the name of the transition, R is a (possibly empty) set of clocks to reset (the update part) and l_j is the end location. A location is *urgent* when the automaton must not remain in the location.

Encoding CSTNUs into TGAs. In what follows, we provide the encoding given in (Cimatti *et al.*, 2016) from a CSTNU into an equivalent TGA. Since we use UPPAAL-TIGA for checking the resulting TGA, we use integer and Boolean variables to improve efficiency and readability.

Assume that $\mathcal{G}$ in Fig. 3 is the TGA equivalent to the CSTNU $\mathcal{S}$ in Fig. 2. The core of $\mathcal{G}$ consists of three locations:

- `ctrl` (urgent) modeling the execution of the non contingent time points,
- `env` (initial) modeling the occurrence of contingent time points and truth value assignments, and
- `goal`, which, if reached, implies DC of $\mathcal{S}$.

Differently from the traditional use of TGAs where `env` is associated with uncontrollable transitions, here `env` is assigned with *controllable* transitions, whereas `ctrl` is assigned with *uncontrollable* ones. This is necessary to model the fact that in any CSTNU `env` must be able to react instantaneously upon the execution of a control time point (Cimatti *et al.*, 2016).

We model the interplay of the game between `ctrl` and `env` by means of a clock c_δ and two transitions,

- $(\texttt{ctrl}; \top; \texttt{pass}; c_\delta := 0; \texttt{env})$, and
- $(\texttt{env}; c_\delta > 0; \texttt{gain}; \emptyset; \texttt{ctrl})$

The first transition guarantees that, after an action of the agent, the environment can react immediately. The second one guarantees that the agent can react to an environment action only after a positive delay. Each control time point X is associated to a clock cX and a Boolean variable xX (initially set to $\bot$). We model the execution of X as an *uncontrollable* self-loop transition

- $(\texttt{ctrl}; \neg xX; \texttt{ExX}; xX := \top, cX := 0; \texttt{ctrl})$.

Fig. 3 contains 4 transitions shortened as τ_{A_1}, $\tau_{U?}$, τ_{A_2}, τ_{A_4} modeling the execution $A_1, U?, A_2, A_4$ of Fig. 2.

Clock $\hat{c}$ represents the global time and it is never reset. $\neg xX$ (equivalently, $cX = \hat{c}$) means that X has not been executed yet, whereas xX (equivalently, $cX < \hat{c}$) means that X has been executed. Since cX will be never reset again and clocks never stop, the difference $\hat{c} - cX$ represents the execution time instant of X, when greater than 0, as the run starts at `env` and gets to `ctrl` only after a positive amount of time (`gain` transition, $c_\delta > 0$).

Each propositional letter p associated to an observation control time point $P?$ has an associated bounded integer variable `p` whose value is 0 before $P?$ executes (i.e., for p unknown), -1 if p is assigned $\bot$, and 1 if p is assigned $\top$. To manage the truth value assignment to p, we have two controllable transitions

- $(\texttt{env}; xP? \wedge \texttt{p} = 0; \texttt{pTrue}; \texttt{p} := 1, c_\delta := 0; \texttt{env})$, and
- $(\texttt{env}; xP? \wedge \texttt{p} = 0; \texttt{pFalse}; \texttt{p} := -1, c_\delta := 0; \texttt{env})$

setting $\texttt{p} = 1$ and $\texttt{p} = -1$, respectively. A third uncontrollable transition

- $(\texttt{env}; xP? \wedge \texttt{p} = 0 \wedge c_\delta > 0; \texttt{fail}_\texttt{p}; \emptyset; \texttt{goal})$

allows `ctrl` to win the game (i.e., to get to `goal`) if `env` fails or refuses to assign p a value immediately after $P?$ has been executed. In Fig. 3, we have exactly 3 transitions for u (the unique proposition appearing in the CSTNU in Fig. 2) shortened as $\tau^{\texttt{u}}_{\bot}$, $\tau^{\texttt{u}}_{\top}$ (truth value assignment), and $\phi_{\texttt{u}}$ (fail transition).

For each contingent link $(A, x, y, C) \in \mathcal{L}$, `env` has a controllable self-loop transition

- $(\texttt{env}; xA \wedge \neg xC \wedge cA \geq x \wedge cA \leq y; \texttt{ExC}; xC := \top, cC := 0, c_\delta := 0; \texttt{env})$

to execute C, and `ctrl` has an *uncontrollable* one

- $(\texttt{ctrl}; xA \wedge \neg xC \wedge cA > y; \texttt{fail}_C; \emptyset; \texttt{goal})$

to win the game if env fails or refuses to schedule C such that $C - A \in [x, y]$. The first transition states that the activation point has been executed (xA), C has not been yet executed ($\neg xC$) and it is still within the allowed bounds x and y, since cA, representing the time since the execution of A, is greater-or-equal than x and lower-or-equal than y. *Fail transitions* (for any uncertain part) are necessary, otherwise env would always be able to prevent ctrl from winning the game just "waiting forever" in its own location. In Fig. 3, we have 6 transitions shortened as τ_{C_1}, τ_{C_2}, τ_{C_4} (contingent point executions), and ϕ_{C_1}, ϕ_{C_2}, ϕ_{C_4} (fail).

We model *winning conditions* as a winning path of $n+1$ urgent locations $L_\square \to L_{\ell_1} \to \cdots \to L_{\ell_n}(=\texttt{goal})$ going from env to goal, where n is the number of distinct labels appearing in $\mathcal{S}$. The first location in this path is always $L_\square$, reachable as soon as all "unlabeled" time points have been executed satisfying all related constraints. Then, the remaining locations are sequentially connected through sets of transitions.

Each set of transitions between two consecutive locations represents two cases: (i) the first location is not associated to the current scenario and a transition allows one to move to the next location, or (ii) the first location is associated to the current scenario and a transition allows one to move to the next location only if all the constraints labeled by the label the next location represents are satisfied.

For example, consider the set of transitions going from L_u to goal (i.e., $L_{\neg u}$) in Fig. 3. The related conditional constraint to generate this set is: "in the scenario in which $U?$ has been executed and u has been assigned $\perp$, all time points labeled by $\neg u$ must be executed and all constraints labeled by $\neg u$ must be satisfied". This condition is expressed by: $(xU? \wedge u = -1) \Rightarrow \texttt{G}_{\neg u}$, where $\texttt{G}_{\neg u} \stackrel{def}{=} xA_4 \wedge xC_4 \wedge cU? - cA_4 \leq 5 \wedge cA_4 - cU? \leq -1 \wedge cA_1 - cA_4 \leq 30 \wedge cA_4 - cA_1 \leq 0$.[2] Since TGAs do not allow disjunction nor negation (of clock constraints) in the guards, we rewrite the condition as $\neg(xU? \wedge u = -1) \vee \texttt{G}_{\neg u}$, which simplifies to $\neg xU? \vee u = 1 \vee \texttt{G}_{\neg u}$, and we generate a transition for each disjunct (Fig. 3, $\texttt{sat}_{\neg u}$, $\texttt{skip}^1_{\neg u}$, $\texttt{skip}^2_{\neg u}$). We proceed similarly to connect $L_\square$ to L_u.

DC checking is done by model checking the resulting TGA and seeking a control strategy for env to win the game. In UPPAAL-TIGA, this goal is formalized as `control: A[] not tga.goal`. If UPPAAL-TIGA answers `Property is NOT satisfied`, then the network is DC and a counter-strategy for ctrl can be printed out.

[2]For each $Y - X \leq k \in \mathcal{C}$, the corresponding guard is $(\hat{c} - cY) - (\hat{c} - cX) \leq k$, which simplifies in $cX - cY \leq k$ as $\forall X \in \mathcal{T}, (\hat{c} - cX)$ is the time in which X was executed.

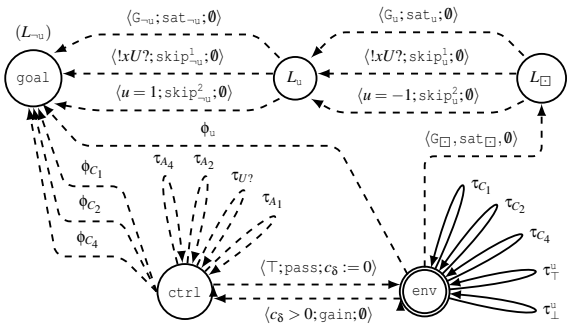

Figure 3: TGA equivalent to the CSTNU in Fig. 2.

4 ACTNs

An ACTN extends a CSTNU by adding a set of users $\mathcal{U}$ and turning $\mathcal{C}$ into a set of sets each one containing (possibly several disjunctive) *authorization constraints*. Users are in charge of executing time points, whereas authorization constraints say which users are authorized to execute the time points they connect. Before giving the formal definition, we introduce a few useful notions.

A *relation* over users is denoted by $\rho \subseteq \mathcal{U} \times \mathcal{U}$, with the *neutral relation* $* \stackrel{def}{=} \mathcal{U} \times \mathcal{U} \stackrel{def}{=} \{(u_1, u_2) \mid u_1 \in \mathcal{U} \wedge u_2 \in \mathcal{U}\}$ and the complement of ρ denoted by $\neg\rho$. If $(u_1, u_2) \in \rho$, then we say that the pair *satisfies* the relation.

Given a finite set $\mathcal{R} = \{\rho_1, \ldots, \rho_n\}$ of relations, an *authorization policy* α is any conjunction of relations drawn from $\mathcal{R}$, each one appearing as ρ or $\neg\rho$ if different from $*$. If a relation appears as ρ (respectively, $\neg\rho$), then we say that ρ is *positive* (respectively, *negative*). The neutral authorization policy, i.e., that consisting of $*$ only, is denoted by $\circledast$. A pair (u_1, u_2) *satisfies* an authorization policy α iff (u_1, u_2) satisfies each $\rho \in \alpha$ and does not satisfy each ρ such that $\neg\rho \in \alpha$. An authorization policy α_1 *entails* another authorization policy α_2 (written $\alpha_1 \Rightarrow \alpha_2$) iff α_1 contains all the relations appearing in α_2. Two authorization policies α_1 and α_2 are *consistent* if $\alpha_1 \wedge \alpha_2$ is satisfiable, and *equivalent* if both α_1 entails α_2 and α_2 entails α_1 (i.e., $\alpha_1 \Rightarrow \alpha_2$ and $\alpha_2 \Rightarrow \alpha_1$).

For example, suppose that $\mathcal{U} = \{\texttt{John}, \texttt{Lara}, \texttt{Tom}, \texttt{Bob}, \texttt{Sara}, \texttt{Rick}, \texttt{Kate}\}$ and $\mathcal{R} = \{*, \neq, \textit{Relatives}\}$, with $\neq \stackrel{def}{=} \{(u_1, u_2) \mid u_1, u_2 \in \mathcal{U} \wedge u_1 \neq u_2\}$ and $\textit{Relatives} \stackrel{def}{=} \{(\texttt{John}, \texttt{Kate}), (\texttt{Kate}, \texttt{John})\}$. Consider the policy $\alpha \stackrel{def}{=} \neq \wedge \neg \textit{Relatives}$ given in Fig. 1 between PatEv and LTTh saying that the physician giving the patient the long term therapy must be different

and not a relative of the physician who did the initial evaluation. If John carries out the initial PatEv, then Lara can execute LTTh as (John, Lara) satisfies α.

Definition 3. An *Access-Controlled Temporal Network (ACTN)* is a tuple $\langle \mathcal{T}, \mathcal{U}, UA, \mathcal{C}, \mathcal{R}, L, O\mathcal{T}, O, \mathcal{P}, \mathcal{CT}, \mathcal{L} \rangle$, where:

- $\mathcal{T}, L, O\mathcal{T}, O, \mathcal{P}, \mathcal{CT}, \mathcal{L}$ are the same as for a CSTNU along with all assumptions on contingent links (A, x, y, C).
- $\mathcal{U}$ is a non-empty finite *set of users*.
- $UA \subseteq \mathcal{U} \times \mathcal{T}$ is the *authorization relation*. $\mathcal{A}(X) = \{u \mid (u, X) \in UA\}$ is the *set of users authorized for* X.
- $\mathcal{R}$ is a finite set of relations $\rho \subseteq \mathcal{U} \times \mathcal{U}$.
- $\mathcal{C}$ is a set of sets such that each set $\mathcal{C}_{XY} \in \mathcal{C}$ consists of a (possibly many) authorization constraints of the form $(x \leq Y - X \leq y, \ell : \alpha)$[3] where $x, y \in \mathbb{R} \cup \pm\infty$, $Y, X \in \mathcal{T}$, $\ell \in \mathcal{P}^*$, α is an authorization policy.
- $\ell = L(X) \wedge L(Y)$ for each $(x \leq Y - X \leq y, \ell : \alpha) \in \mathcal{C}_{XY}$, and $\ell, L(X), L(Y)$ entail $L(P?)$ for each $p, \neg p \in \ell, L(X), L(Y)$.
- For each pair $(x_1 \leq Y - X \leq y_1, \ell_1 : \alpha_1), (x_2 \leq Y - X \leq y_2, \ell_2 : \alpha_2)$ belonging to the *same* set $\mathcal{C}_{XY}$, if $[x_1, y_1] = [x_2, y_2]$, then $\alpha_1 \neq \alpha_2$ (note that $\ell_1 = \ell_2$).
- For each $(A, x, y, C) \in \mathcal{L}$, we have that A is non-contingent (i.e., $A \notin \mathcal{CT}$), and $\mathcal{A}(A) = \mathcal{A}(C)$.
- For each $(A_1, x_1, y_1, C_1), (A_2, x_2, y_2, C_2) \in \mathcal{L}$, we have that $A_1 \not\equiv A_2$ and $C_1 \not\equiv C_2$.

The *ACTN-graph* extends the CSTNU-graph by labeling requirement links $X \to Y$ by (possibly many) labels of the form $[x, y], \ell : \alpha$ each one corresponding to the authorization constraint $(x \leq Y - X \leq y, \ell : \alpha) \in \mathcal{C}_{XY}$. Fig. 4 gives an example of an ACTN that models the workflow in Fig. 1 considering access control. Again, we map PatEv to $(A_1, 5, 25, C_1)$ and its authorized users are $\mathcal{A}(A_1) = \mathcal{A}(C_1) = \{$John, Lara$\}$ who belong to the role EmergencyDoctor. We map SurInt and ICUStayM to $(A_2, 60, 210, C_2)$ and $(A_3, 2, 4, C_3)$, respectively, and their authorized users are $\mathcal{A}(A_2) = \mathcal{A}(C_2) = \mathcal{A}(A_3) = \mathcal{A}(C_3) = \{$Bob, Sara, Tom$\}$, who belong to the role Surgeon. We map FibTh, ThAst, and LTTh to $(A_4, 60, 65, C_4)$, $(A_5, 2, 10, C_5)$, and $(A_6, 5, 10, C_6)$, respectively, and their authorized users are $\mathcal{A}(A_4) = \mathcal{A}(C_4) = \mathcal{A}(A_5) = \mathcal{A}(C_5) = \mathcal{A}(A_6) = \mathcal{A}(C_6) = \{$Lara, Kate, Rick$\}$, who belong to the role Cardiologist (note that Lara is a specialist in two medical fields, so she belongs to two roles).

[3] $(x \leq Y - X \leq y, \ell : \alpha)$ shortens $(Y - X \leq y, \ell : \alpha)$ and $(X - Y \leq -x, \ell : \alpha)$. We assume $\mathcal{C}_{XY} = \mathcal{C}_{YX}$.

We model the conditional split and join connectors (shaded diamonds) by means of $U?$ and E, and we connect all the components by means of authorization constraints, which also impose relative constraints among two non-sequential tasks specifying authorization policies. For example, we model the TSoD between tasks SurInt and ICUStayM by means of the authorization constraint $C_2 \xrightarrow{[1,3],u:\neq} A_3$. It says that once C_2 has been executed (i.e., SurInt has been completed), say by Bob at time $t = 90$, then either Tom or Sara can execute A_3 (i.e., start ICUStayM) from time 91 to 96. The second authorization constraint $C_2 \xrightarrow{[3+\varepsilon,6],u:\circledast} A_3$ ensures that TSoD does not apply otherwise allowing all users to execute A_3 from time $93 + \varepsilon$ to 96 (we use ε to make sure that no overlap exists between $[1, 3]$ and $[3 + \varepsilon, 6]$).

Execution Semantics. We give the execution semantics of ACTNs in terms of *real-time execution decisions (RTEDs)*. Intuitively, an RTED is a decision to commit some users(s) to executing a set of time points, or a decision to wait for something to happen or an instantaneous reaction.

In what follows, we adapt the RTEDs for CSTNUs given in (Cimatti *et al.*, 2016) by also renaming a few symbols and adding details for explanatory purposes. For an ACTN, the *controller* (ctrl) seeks a strategy for committing available users to executing all relevant control time points (i.e., the part under control) such that all relevant constraints in $\mathcal{C}$ will eventually be satisfied no matter what durations the *environment* (env) chooses for contingent links and truth values for propositions. Thus, RTEDs exist for both ctrl and env. We study the interplay between these two RTEDs in terms of partial and full outcomes, which determine how the state of the whole system evolves.

In this paper, we assume that each contingent link (A, x, y, C) is (i) tied to one range $[x, y]$ only, and (ii) executed by one user only who, once he has executed A, remains blocked until the execution of C. We leave as future work the generalization of the approach to handling situations in which users are given less (or more) time for the same task.

The *set of all available users* is $Avail \subseteq \mathcal{U}$, so that $u \in Avail$ means that u is available and busy otherwise. The initial state is $Avail = \mathcal{U}$ meaning that all users are available. Blocking users guarantees us that the same user can be committed for more than one task in a parallel block ensuring this user will be always executing one task at a time.

Definition 4. A *partial schedule* for an ACTN is a pair $\mathcal{PS} = (OccTP, KnownProp)$, where $OccTP$ is a set of triples (u, X, k) meaning that user u executed time point X at time k, and $KnownProp$ is a set of pairs

(p,b) meaning that p has been assigned $b \in \{\top,\bot\}$.

We represent the *current partial scenario* (i.e., the scenario being generated upon the execution of the observation time points) by means of the label ℓ_{cps} consisting of the conjunction of the already known literals (i.e., $\ell_{cps} = \{p \mid (p,\top) \in \mathit{KnownProp}\} \cup \{\neg q \mid (q,\bot) \in \mathit{KnownProp}\}$).

$\mathit{Executed}(\mathit{OccTP}) = \{X \mid (u,X,k) \in \mathit{OccTP}\}$ is the *set of executed time points* and $\mathit{Instants}(\mathit{OccTP}) = \{k \mid (u,X,k) \in \mathit{OccTP}\}$ is the *set of time instants in which the time points in* $\mathit{Executed}(\mathit{OccTP})$ *were executed*. For each $X \in \mathit{Executed}(\mathit{OccTP})$, we represent the time instant k in which X was executed as $\mathit{time}(X)$ and the user who executed it as $\mathit{user}(X)$.

We represent the time instant of the last executed time point as $\mathit{last}(\mathit{OccTP}) = \max\{v \mid v \in \mathit{Instants}(\mathit{OccTP})\}$; if $\mathit{OccTP} = \emptyset$, then $\mathit{last}(\mathit{OccTP}) = -\infty$.

$\mathcal{PS}$ is called *respectful* if all $(u,X,k) \in \mathit{OccTP}$ satisfy the constraints in $\mathcal{C}$. The set of all possible partial schedules is represented by $\mathcal{PS}^*$.

Definition 5. An *RTED for* `ctrl` has two forms: `wait` or $(t, \mathit{ControlTP})$.

- $\Delta_{\texttt{ctrl}} = \texttt{wait}$ is applicable only if a contingent time point has been activated (i.e., the activation time point A has been executed but the related contingent C has not).
- $\Delta_{\texttt{ctrl}} = (t, \mathit{ControlTP})$ represents the conditional constraints "if `env` does nothing before time t, then for each pair $(u,X) \in \mathit{ControlTP}$, commit the user u to execute the time point X at time t". Such an RTED is applicable iff $t > \mathit{last}(\mathit{OccTP})$, $\mathit{ControlTP}$ is a non empty set of pairs of available users and unexecuted time points. That is, for each $(u,X) \in \mathit{ControlTP}$, $X \notin \mathit{Executed}(\mathit{OccTP})$ and $u \in \mathit{Avail}$. Finally, for each $(u_1,X_1),(u_2,X_2) \in \mathit{ControlTP}$, if X_1 and X_2 are (different) activation points, then $u_1 \neq u_2$. If X_1 is an activation point and X_2 is not, then we execute X_2 first. This is because activation points block users.

An *RTED for the environment* `env` (in symbols, $\Delta_{\texttt{env}}$) has two forms: `wait` or $(t, \mathit{ContingentTP})$.

- $\Delta_{\texttt{env}} = \texttt{wait}$ is applicable only if no contingent point C has been activated.
- $\Delta_{\texttt{env}} = (t, \mathit{ContingentTP})$ represents the conditional constraints "if `ctrl` does nothing before or at time t, then execute the time points in $\mathit{ContingentTP}$ at time t". Such a decision is applicable iff $t > \mathit{last}(\mathit{OccTP})$, $\mathit{ContingentTP}$ is a non empty subset of unexecuted contingent points whose ranges of allowed durations contain t.

The *sets of all RTEDs for* `ctrl` *and* `env` are represented by $\Delta^*_{\texttt{ctrl}}$ and $\Delta^*_{\texttt{env}}$, respectively.

In other words, $\Delta_{\texttt{ctrl}}$ deals with the parts under control: users and non-contingent time points. $\Delta_{\texttt{env}}$ deals with the parts out of control: contingent time points and truth value assignments.

Definition 6. Let $\mathcal{PS} = (\mathit{OccTP}, \mathit{KnownProp})$ be a partial schedule in which at least one contingent time point C has been activated and is allowed to be executed at $\mathit{last}(\mathit{OccTP})$. An *instantaneous reaction* $I\mathcal{R}$ is a decision

(1) to execute a set I_C of such time points at time $\mathit{last}(\mathit{OccTP})$, or
(2) to assign truth values for each proposition associated to the observation time points in OccTP that has not been assigned yet (we model such an assignment as a set I_B of pairs (p,b) with $p \in \mathcal{P}$ and $b \in \{\top,\bot\}$), or
(3) to do both actions.

We represent an $I\mathcal{R}$ as a pair (I_C, I_B). The *set of all instantaneous reactions* is represented by $I\mathcal{R}^*$.

If $\mathit{last}(\mathit{OccTP})$ happens to be the last possible time at which a currently-activated contingent time point C can execute, then the instantaneous reaction must include C. Similarly, if t is the time in which an observation time point P? was executed, the instantaneous reaction must include a truth value assignment for p.

Of course, `env` can carry out more than one $I\mathcal{R}$. We now define how $\Delta_{\texttt{ctrl}}$ and $\Delta_{\texttt{env}}$ are handled. Since both players can either wait or conditionally commit to executing a set of time points, we have four possible cases (we point out that a `wait` decision is not applicable for both `ctrl` and `env` simultaneously).

Definition 7. We model the *partial outcome* of $\Delta_{\texttt{ctrl}}$ and $\Delta_{\texttt{env}}$ as $O_p(\mathit{OccTP}, \Delta_{\texttt{ctrl}}, \Delta_{\texttt{env}})$ neglecting any $I\mathcal{R}$. We have four possible cases.

(1) $O_p(\mathit{OccTP}, \texttt{wait}, (t, \mathit{ContingentTP})) = \mathit{OccTP} \cup \{(\mathit{user}(A), C, t) \mid C \in \mathit{ContingentTP}\}$ and for each $C \in \mathit{ContingentTP}, \mathit{Avail} = \mathit{Avail} \cup \mathit{user}(C)$, where A is the activation time point that activated C.
(2) $O_p(\mathit{OccTP}, (t_1, \mathit{ControlTP}), (t_2, \mathit{ContingentTP})) = \mathit{OccTP} \cup \{(\mathit{user}(A), C, t_2) \mid C \in \mathit{ContingentTP}\}$ if $t_2 < t_1$ and for each $C \in \mathit{ContingentTP}, \mathit{Avail} = \mathit{Avail} \cup \mathit{user}(C)$, where A is the activation time point that activated C.
(3) $O_p(\mathit{OccTP}, (t, \mathit{ControlTP}), \texttt{wait}) = \mathit{OccTP} \cup \{(u,X,t) \mid (u,X) \in \mathit{ControlTP}\}$ and for each $(u,A) \in \mathit{ControlTP}, \mathit{Avail} = \mathit{Avail} \setminus u$.
(4) $O_p(\mathit{OccTP}, (t_1, \mathit{ControlTP}), (t_2, \mathit{ContingentTP})) = \mathit{OccTP} \cup \{(u,X,t_1) \mid (u,X) \in \mathit{ControlTP}\}$ if $t_1 \leq t_2$ and for each $(u,A) \in \mathit{ControlTP}, \mathit{Avail} = \mathit{Avail} \setminus u$.

For example, $O_p(\mathit{OccTP}, (0, \{(\texttt{John}, A_1)\}), \texttt{wait}) = \mathit{OccTP} \cup \{(\texttt{John}, A_1, 0)\}$.

(1) says that `env` can execute the time points in $\mathit{ContingentTP}$ at time t if `ctrl` decides to do nothing

at that time. (2) says that env can execute the time points in *ContingentTP* if he decided to do so before ctrl executes his. (3) is similar to (1) but with respect to ctrl. (4) says that when ctrl decides to execute a set of time points before or at the same time of those env has decided to execute, ctrl moves first to allow env to react instantaneously (Definition 8).

Definition 8. We model the *full outcome* of $\Delta_{\texttt{ctrl}}$ and $\Delta_{\texttt{env}}$ as $O(OccTP, \Delta_{\texttt{ctrl}}, \Delta_{\texttt{env}}, I\mathcal{R})$ and define it as we did for $O_p(OccTP, \texttt{wait}, (t, ContingentTP))$ except that in cases (3) and (4) *OccTP* is augmented with $\{(C,t) \mid C \in I_C\}$ (applicable if C has been activated), and *KnownProp* is augmented with $\{(p,b) \mid (p,b) \in I_B\}$ (applicable if P? has been executed). Either way $t = last(OccTP)$.

The full outcome says how the state of the system (i.e., $\mathcal{PS}$) evolves according to the interplay of ctrl's and env's RTEDs.

Definition 9. An *RTED-based strategy for* ctrl is a mapping $\sigma_{\texttt{ctrl}} : \mathcal{PS}^* \to \Delta^*_{\texttt{ctrl}}$ from respectful partial schedules to RTEDs. An *RTED-based strategy for* env is a pair of mappings $\sigma_{\texttt{env}} = (\mu_{\Delta_{\texttt{env}}}, \mu_{IR})$, where $\mu_{\Delta_{\texttt{env}}} : \mathcal{PS}^* \to \Delta^*_{\texttt{ctrl}}$ is a mapping from respectful partial schedules to RTEDs, and $\mu_{IR} : \mathcal{PS}^* \to I\mathcal{R}^*$ is a mapping from respectful partial schedules to instantaneous reactions.

Definition 10. The *one-step outcome* of the game modeling the execution of the network by ctrl and env is defined as $O^1(OccTP, \sigma_{\texttt{ctrl}}, \sigma_{\texttt{env}}) = O(OccTP, \sigma_{\texttt{ctrl}}(OccTP), \mu_{\Delta_{\texttt{env}}}(OccTP), \mu_{IR}(OccTP_p))$, where $OccTP_p = O_p(OccTP, \sigma_{\texttt{ctrl}}(OccTP), \mu_{\Delta_{\texttt{env}}}(OccTP))$.

The *terminal outcome* $O^*(\sigma_{\texttt{ctrl}}, \sigma_{\texttt{env}})$ is the complete schedule that results from the recursive definition: $OccTP_0 = \emptyset$, $OccTP_{i+1} = O^1(OccTP_i, \sigma_{\texttt{ctrl}}, \sigma_{\texttt{env}})$.

Definition 11. An ACTN is DC if there exists an RTED-based strategy $\sigma_{\texttt{ctrl}}$ such that for all RTED-based strategies $\sigma_{\texttt{env}}$, the variable assignments (u, X, k) in the complete schedule $O^*(\sigma_{\texttt{ctrl}}, \sigma_{\texttt{env}})$ satisfy all constraints in $\mathcal{C}$.

5 DYNAMIC CONTROLLABILITY OF ACTNS USING UPPAAL-TIGA

In this section, we extend the mapping given in § 3, we use UPPAAL-TIGA as an off-the-shelf model checker and we discuss a few optimizations.

Assume that $\mathcal{G}$ in Fig. 5 is the TGA equivalent to the ACTN $\mathcal{S}$ in Fig. 4. The core of the TGA remains the same as that discussed in § 3.

Internal State. We represent relations, users and their availability by extending the internal states of the TGA with a new piece of information. We assign a unique incremental integer starting from 0 to each time point, and proceed similarly for each user and proposition. Mapping time points, users and propositions to integers allows us to employ them as indexes over array data structures we are going to use in UPPAAL-TIGA. Without loss of generality, when intended as indexes, we abuse notation and write u, X, and p meaning their assigned integer.

We chose to model availability of users as a Boolean vector *Avail* indexed on users. Hence, $Avail[u] = \top$ means that u is available and busy otherwise. The initial state of *Avail* consists of all elements set to $\top$ meaning that all users are available.

We keep track of *who did what* by means of a system trace vector *SysTrace* whose index ranges over time points. If $SysTrace[X] = u$, then time point X has been executed by user u. The initial state of *SysTrace* consists of all elements set to -1, meaning that all time points have not been executed yet.

For instance, if John has executed A_1, then $SysTrace[A_1] =$ John. We do not need to keep track of *when* a time point X has been executed since its value is given by the difference between global time and its clock.

We reorganize clocks associated to time points by means of a vector *clk* indexed on time points, i.e., $clk[X]$ is the clock associated to the time point X. The vector *clk* does not contain clocks $\hat{c}$ (representing global time) and c_δ (used in the guards and updates of the transitions modeling the interplay between ctrl and env). Likewise, we reorganize Boolean variables associated to time points by means of a vector *xtd* indexed on time points whose elements are initialized to $\bot$. Finally, we reorganize propositions by means of an integer vector *prop* whose elements are initialized to 0.

Predecessors and Transition Range Limitations. Before extending the transition guards and updates we discuss a few optimizations. The mapping given in § 3 generates a transition for each time point. However, if we only focus on transitions involving control time points (i.e., those representing the execution of all X such that $X \notin \mathcal{CT}$), one of the problems is that there is no partial order for these transitions.

As a results, that approach suffers from a full state explosion as we can take those transitions to generate impossible runs. To give an example, consider the workflow in Fig. 1. Assume that the patient is

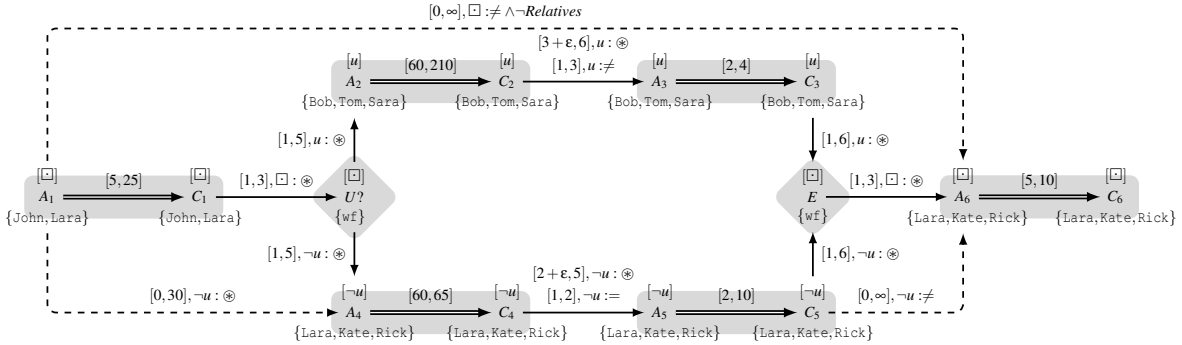

Figure 4: ACTN modeling the workflow given in Fig. 1.

not urgent (i.e., $u = \bot$). In this scenario, we expect to execute PatEv *then* SurInt *then* ICUStayM, and *finally* LTTh. However, the transitions for A_1, A_2, A_3, A_6 (modeling to the start of the tasks just mentioned) also allow us to explore those runs violating this execution order (e.g., SurInt before PatEv or LTTh before SurInt). We enforce the execution order among time points in the guards of the corresponding transitions only when we are sure this order is not ambiguous.

Definition 12. Let Y be a control time point ($Y \notin \mathcal{CT}$). For any $X \in \mathcal{T}$ different from Y, we say that X is a *predecessor* of Y (and write $X \in \Pi(Y)$) if all authorization constraints between X and Y have a positive lower bound ($x \geq 0$) and label ℓ identical to that of Y. Formally:

$$\Pi(Y) = \{X \mid (x \leq Y - X \leq y, \ell : \alpha) \in \mathcal{C}_{XY} \wedge x \geq 0 \wedge \ell = L(Y)\}$$

If $X \in \Pi(Y)$ it follows that $Y - X \in [x_{min}, y_{max}]$, where

- $x_{min} = \min\{x \mid (x \leq Y - X \leq y, \ell : \alpha) \in \mathcal{C}_{XY}\}$, and
- $y_{max} = \max\{y \mid (x \leq Y - X \leq y, \ell : \alpha) \in \mathcal{C}_{XY}\}$

That is, if all authorization constraints between X and Y have a positive lower bound and label equal to $L(Y)$ (i.e., we must satisfy at least one of them whenever we consider Y), then Y definitely executes after X. Moreover, once we have executed X, we can execute Y after at least the minimum among the lower bounds (x_{min}) and within the maximum of the upper bounds (y_{max}) of the authorization constraints in $\mathcal{C}_{XY}$, to consider all the possible disjunctions.

As an example, consider Fig. 4. $\Pi(A_3) = \{C_2\}$ since $\mathcal{C}_{C_2A_3} = \{(1 \leq A_3 - C_2 \leq 3, u :\neq), (3+\varepsilon \leq A_3 - C_2 \leq 6, u : \circledast)\}$ and $L(A_3) = u$. Moreover, once env executes C_2, ctrl can execute A_3 such that $A_3 - C_2 \in [1,6]$ ($x_{min} = 1$ and $y_{max} = 6$). Likewise, $\Pi(A_5) = \{C_4\}, A_5 - C_4 \in [1,5]$. $\Pi(U?) = \{C_1\}$ and $U? - C_1 \in [1,3]$. $\Pi(A_6) = \{E, A_1\}, A_6 - E \in [1,3]$ and $A_6 - A_1 \in [0,\infty]$. $\Pi(A_2) = \{U?\}$ and $A_2 - U? \in [1,5]$. $\Pi(A_4) = \{U?, A_1\}$ and $A_4 - U? \in [1,5], A_4 - A_1 \in [0,30]$.

Instead, we cannot do much for E, since each constraint having E as a target has a label that is not entailed by $\boxdot$ (i.e., $L(E)$). That is, for $(1 \leq E - C_3 \leq 6, u : \circledast) \in \mathcal{C}_{C_3E}$ and $(1 \leq E - C_5 \leq 6, \neg u : \circledast) \in \mathcal{C}_{C_5E}$, we have that $L(E) \not\Rightarrow u$ nor $L(E) \not\Rightarrow \neg u$. We recall that we always execute E since its label, $\boxdot$, is entailed by any scenario. But, if scenario u (respectively, $\neg u$) holds, the guard of the transition enforcing the partial order (i.e., that considering $\Pi(E)$) would become conditional depending on the current partial scenario. For E, we would generate three transitions: one for $\boxdot$ with $\Pi(E) = \emptyset$, one for u with $\Pi(E) = \{C_3\}$ and $C_3 - E \in [1,6]$, and another for $\neg u$ with $\Pi(E) = \{C_5\}$ and $C_5 - E \in [1,6]$.

Although such an approach risks making the encoding exponential, we should also prove that the generated transitions do not prevent some scenario from being explored by the model-checking phase. For this reason, each $\Pi(Y)$ contains a predecessor only when we are sure that $X \in \Pi(Y)$ always comes before Y (similarly, we do not consider the case "C_5 is before A_6").

Note that activation time points are trivially before their related contingent and already handled with a similar approach in the guards of the transitions executing them at env.

Time Point Transitions. For each user authorized for X (where $X \notin \mathcal{CT}$), we have an uncontrollable self-loop transition having the form:

$$(\texttt{ctrl}; \mathit{Guard}(u,X); \texttt{uExX}; \mathit{Update}(u,X,\mathit{setBusy}); \texttt{ctrl})$$

where:

- $\mathit{Guard}(u,X) \stackrel{def}{=} \neg \mathit{xtd}[X] \bigwedge_{p \in L(X)} (\mathit{prop}[p] = 1) \bigwedge_{\neg q \in L(X)} (\mathit{prop}[q] = -1) \wedge \mathit{Avail}[u] \bigwedge_{Y \in \Pi(X)} (\mathit{xtd}[Y] \wedge \mathit{clk}[Y] \geq x_{min} \wedge \mathit{clk}[Y] \leq y_{max})$. This function formalizes the mandatory part in the guard: X has not been executed yet, $L(X)$ is true, the authorized user u is available, all $Y \in \Pi(X)$ have been executed and $\mathit{clk}[Y] \in [x_{min}, y_{max}]$ (as discussed before).

- $Update(u,X,setBusy) \stackrel{def}{=} xtd[X] := \top$, $clk[X] := 0$, $SysTrace[X] := u$, $Avail[u] := \neg setBusy$. This function formalizes the mandatory part in the update: X has been executed by u at time $\hat{c} - clk[X]$, and u is now busy ($setBusy$) if X is an activation time point.

For concreteness, consider the self loop $\tau_{\texttt{Kate},A_5} = (\texttt{ctrl}; Guard(\texttt{Kate},A_5); \texttt{KateExA5}; Update(\texttt{Kate},A_5, \top); \texttt{ctrl}\rangle$ in Fig. 5. It says that if the patient is not urgent, `Kate` can start `ThAst` (i.e., `Kate` executes A_5) after 1 and within 5 minutes, after `FibTh` has been completed (i.e., $xtd[C_4] \wedge clk[C_4] \geq 1 \wedge clk[C_4] \leq 5$). $\tau_{\texttt{Kate},A_5}$ along with other 18 similar self loops $\tau_{u,X}$ model the authorized executions of all non contingent time points.

For each contingent time point, we only extend the update part of the transition given in § 3 by adding the two statements $SysTrace[C] := SysTrace[A]$, and $Avail[SysTrace[C]] := \top$. That is, first, we save that the user executing C is the same of that who executed A, and second, we release such a user. Note that the guard does not contain conditions on availability on purpose. Adding such conditions would result in `env` shrinking the range of C.

Game Interplay, Failing Transitions, and Winning Path Considering Disjunctive Constraints. Failing transitions and transitions regulating the game interplay and assigning the truth values to propositions remain the same as those discussed in § 3 (adapted to the new array-reorganization).

The winning path is extended by adding intermediate locations to guarantee that whenever disjunctive authorization constraints exist, we always take into consideration one of them. We proceed by first introducing the set of core constraints (i.e., those we must always consider for the traditional winning path).

Definition 13. Let $\ell \in \mathcal{P}^*$ be a label. The set of *core constraints* labeled by ℓ is given by:

$$Core(\ell) = \bigcup_{C_{XY} \in \mathcal{C}} \{(x \leq Y - X \leq y, \ell : \alpha) \mid |\mathcal{C}_{XY}| = 1 \wedge (x \leq Y - X \leq y, \ell : \alpha) \in \mathcal{C}_{XY} \wedge \ell = L(X) \wedge L(Y)\}$$

For our example, $Core(u) = \{(1 \leq A_2 - U? \leq 5, u : \circledast), (1 \leq E - C_3 \leq 6, u : \circledast)\}$. Note that we do not take into consideration $\mathcal{C}_{C_2A_3} = \{(1 \leq C_3 - C_2 \leq 3, u : \neq), (3+\varepsilon \leq C_3 - C_2 \leq 6, u : \circledast)\}$ as $|\mathcal{C}_{C_2A_3}| = 2$.

$\texttt{G}_\texttt{u}$ in Fig. 5 verifies both that all time points labeled by u have been executed and that the core constraints and their related authorization policies they express are satisfied. $\texttt{skip}_1^\texttt{u}$ and $\texttt{skip}_2^\texttt{u}$ are the same as those discussed for classic CSTNUs in § 3.

We now deal with the disjunctive authorization constraints. We first compute the set of non-core constraints (dual to the set core), i.e., the set containing all the disjunctions with respect to a given label ℓ. Then, we generate the locations and transitions to verify that we satisfy *at least* one authorization constraint for each disjunction avoiding the combinatorial explosion.

Definition 14. Let $\ell \in \mathcal{P}^*$ be a label. The set of *non-core constraints* labeled by ℓ for each $\mathcal{C}_{XY}$ is given by:

$$NonCore(\mathcal{C}_{XY}, \ell) = \{(x \leq Y - X \leq y, \ell : \alpha) \mid |\mathcal{C}_{XY}| > 1 \wedge (x \leq Y - X \leq y, \ell : \alpha) \in \mathcal{C}_{XY} \wedge \ell = L(X) \wedge L(Y)\}$$

For our example, $NonCore(\mathcal{C}_{A_3C_2}, u) = \{(1 \leq A_3 - C_2 \leq 3, u : \neq), (3+\varepsilon \leq A_3 - C_2 \leq 6, u : \circledast)\}$. That is, either A_3 is executed such that $A_3 - C_2 \in [1,3]$ and TSoD ($\neq$) must hold, or A_3 is executed such that $A_3 - C_2 \in [3+\varepsilon, 6]$ and any user can do so ($\circledast$).

To handle the disjunctions with respect to the label ℓ for each $NonCore(\mathcal{C}_{XY}, \ell)$, we create an intermediate location $L_{C_{XY},\ell}$ to avoid generating an exponential number of transitions. For each $L_{C_{XY},\ell}$, we have a set of transitions containing the same `skip` transitions as those for L_ℓ and we have a `sat` transition for each disjunct in $NonCore(\mathcal{C}_{XY}, \ell)$. Concretely, for $NonCore(\mathcal{C}_{A_3C_2}, u)$, we generate $L_{\mathcal{C}_{C_2A_3},u}$ (shortened as $L'_\texttt{u}$ in Fig. 5), $\texttt{sat}^1_{C_2A_3,\texttt{u}}$ (verifying the first disjunction) and $\texttt{sat}^2_{C_2A_3,\texttt{u}}$ (verifying the second one).

Likewise, we generate $L_{\mathcal{C}_{C_4A_5},\neg u}$ (i.e., `goal`), $\texttt{sat}^1_{C_4A_5,\neg\texttt{u}}$ and $\texttt{sat}^2_{C_4A_5,\neg\texttt{u}}$ for $NonCore(\mathcal{C}_{C_4A_5}, \neg u)$.

We discuss the complexity and prove the correctness of the encoding in the appendix (Theorem 1 and Theorem 2-3, respectively).

The ACTN in Fig. 4 is DC. One of the possible executions is the following. `ctrl` starts the workflow in Fig. 1 by assigning `PatEv` to `John`. As soon as `John` finishes (i.e., `env` executes C_1), `ctrl` commits the workflow management system `wf` to execute the conditional split connector (modeled by $U?$). If the patient is urgent (i.e., if u is assigned $\top$), then `SurInt` is assigned to `Tom` and exactly 1 minute after `Tom` is done, `ICUStayM` is assigned to `Bob` (since a TSoD must hold). If the patient is not urgent, `FibTh` and `ThAst` are both carried out by `Lara` because the start of `ThAst` occurs one minute after `FibTh` and thus a TBoD must hold. Regardless of which branch has been taken, `ctrl` commits `wf` to execute the join connector 1 minute after the last task of the chosen branch terminated. Finally, `ctrl` commits `Rick`, who is different from and not related to `Lara` or `John`, to executing `LTTh` 1 minute after the join connector ended.

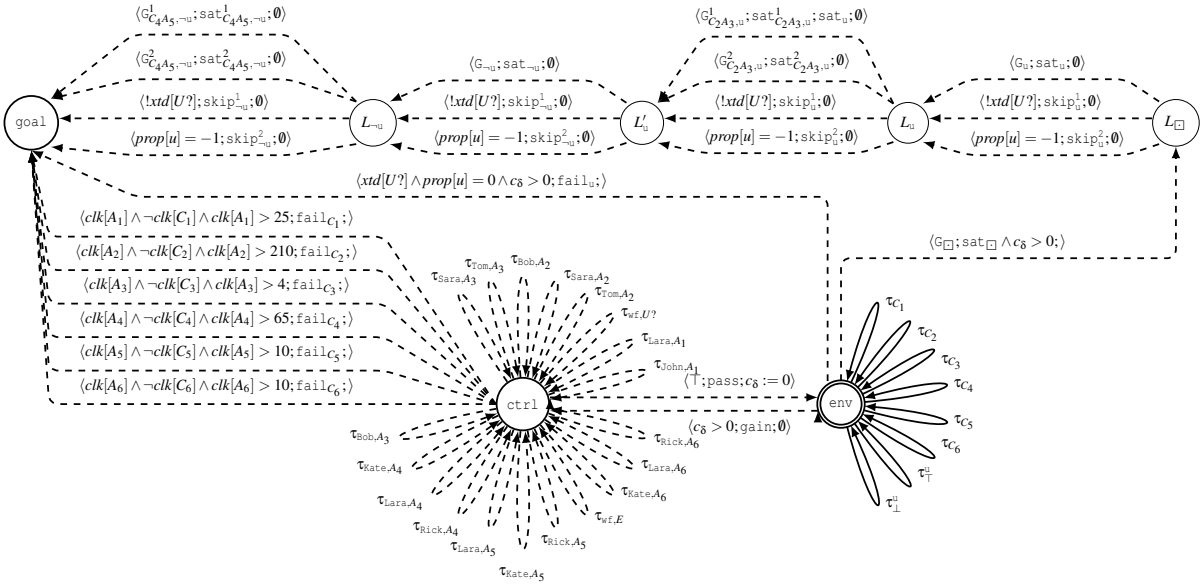

Figure 5: TGA equivalent to the ACTN in Fig. 4.

6 RELATED WORK

There exist two main classes of algorithms for checking the DC of a CSTNU: network-based and TGA-based ones.

The network-based algorithms in (Combi *et al.*, 2013; Combi *et al.*, 2014b) take as input a CSTNU and check if the network is DC: they are sound but not complete.

TGAs allow one to determine a sound and complete DC-checking algorithm. DC of a CSTNU is implemented by first encoding a given CSTNU into an equivalent TGA (as we discussed) and then, seeking a control strategy for the `env`—via TCTL model checking— to prevent `ctrl` from heading to a special location called `goal`. If `env` succeeds, the network is not DC, otherwise it is. The correctness of the approach is proved in (Cimatti *et al.*, 2016).

The structured business process modeling language TNest is introduced in (Combi *et al.*, 2014a). To understand whether a temporal workflow is DC, the TNest specification is translated in an equivalent CSTNU on which DC checking is done. TNest does not deal with users and authorization constraints.

The work closest to ours is by (Combi *et al.*, 2016), who used an STNU (Morris *et al.*, 2001) to model an unconditional temporal role-based access-controlled workflow. They translated a fragment of a TRBAC (Bertino *et al.*, 2001) into a temporal network, and the workflow into another. Then, they connected the resulting networks in order to check if the temporal constraints of the workflow and those of the access-control model are contradictory. Finally, they derived for each task the set of authorized users and gave a formalization of security constraints (SCs) and Security Constraints Propagation Rules (SCPR) to enforce authorization constraints at execution time. DC analysis in the WSP sense is left as future work. In contrast, in this paper we directly inject users and authorization constraints in the formalization of the ACTN itself, and we do not only deal with TSoD and TBoD, but also with other security policies expressed by means of relations over users.

(Wang and Li, 2010) and (Crampton *et al.*, 2014) (to name a few) have addressed the WSP. If an ACTN modeling a workflow is DC, then the temporal WSP is automatically answered. The execution strategy provides us with information about *who and when* has to execute which time points depending on what is going on. This assignment is generated in real time.

Finally, approaches such as those proposed by (Barth *et al.*, 2007) and (Barletta *et al.*, 2011) are not really suitable to model temporal workflows as we are doing. First, we do not need to use messages, and second, we have to know exactly when each transition is taken (with respect to when we started) in order to propagate the effect (which means to run algorithms on the network to update the allowed temporal ranges of the unexecuted time points. Timed computation tree logic (TCTL) is able to deal with this problem (recall that we use dense time, rather than discrete approximations). The logics in (Barth *et al.*, 2007) and (Barletta *et al.*, 2011) have no dense time representation (clocks) and they do not allow for computation

over real-valued temporal constraints.

7 CONCLUDING REMARKS

We defined ACTNs as an extension of CSTNUs in order to take into consideration users and authorization constraints, and we used them to analyze the official STEMI guidelines as a concrete example. We similarly extended the execution semantics for CSTNUs given in terms of RTEDs. After that, we provided an encoding from ACTNs to TGAs discussing a few optimizations to speed up the model-checking phase. We proved that our encoding is polynomial, *fully automated*, and we also showed the correctness (see the appendix). As a result, we provided a *sound* and *complete* approach for the temporal WSP. As a final contribution, we discussed our experimental evaluation using the UPPAAL-TIGA software tool. We are currently implementing the encoder from ACTNs into TGAs and an executor for executing the ACTN following the synthesized strategy.

As future work we will investigate two main directions, distinguished by the use, or not, of TGAs. In the first case, we plan to address the workflow resiliency problem, which is a refinement of the WSP when a subset of the authorized user may become absent before or during the execution (again, a new kind of uncertainty that needs a controllability approach). In the second, we plan to devise a network-based constraint-propagation algorithm for ACTN DC-checking. These algorithms might be more complex but are typically faster than TCTL model checking as they tighten the network by ruling out all impossible execution strategies.

Finally, we plan to provide a structured modeling language for designing access-controlled workflows to be mapped into ACTNs for the DC checking.

REFERENCES

Barletta, M., Ranise, S., and Viganò, L. (2011). A declarative two-level framework to specify and verify workflow and authorization policies in service-oriented architectures. *SOCA*, 5(2):105–137, http://dx.doi.org/10.1007/s11761-010-0073-4.

Barth, A., Mitchell, J., Datta, A., and Sundaram, S. (2007). Privacy and utility in business processes. In *CSF '07*, pages 279–294. http://dx.doi.org/10.1109/CSF.2007.26.

Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K. G., and Lime, D. (2007). Uppaal-tiga: Time for playing games! In Damm, W. and Hermanns, H., editors, *CAV 2007*, LNCS, pages 121–125. http://dx.doi.org/10.1007/978-3-540-73368-3_14.

Bertino, E., Bonatti, P. A., and Ferrari, E. (2001). TRBAC: A temporal role-based access control model. *ACM Trans. Inf. Syst. Secur.*, 4(3).

Cimatti, A., Hunsberger, L., Micheli, A., Posenato, R., and Roveri, M. (2016). Dynamic controllability via timed game automata. *Acta Informatica*, 53(6–8):681–722, http://dx.doi.org/10.1007/s00236-016-0257-2.

Combi, C., Gambini, M., Migliorini, S., and Posenato, R. (2014a). Representing business processes through a temporal data-centric workflow modeling language: An application to the management of clinical pathways. *IEEE Trans. Syst., Man, Cybern., Syst.*, 44(9):1182–1203, http://dx.doi.org/10.1109/TSMC.2014.2300055.

Combi, C., Hunsberger, L., and Posenato, R. (2013). An algorithm for checking the dynamic controllability of a conditional simple temporal network with uncertainty. In *ICAART 2013*, volume 2, pages 144–156. http://dx.doi.org/10.5220/0004256101440156.

Combi, C., Hunsberger, L., and Posenato, R. (2014b). An algorithm for checking the dynamic controllability of a conditional simple temporal network with uncertainty - revisited. In *Agents and Artificial Intelligence*, volume 449 of *CCIS*, pages 314–331. http://dx.doi.org/10.1007/978-3-662-44440-5_19.

Combi, C., Viganò, L., and Zavatteri, M. (2016). Security constraints in temporal role-based access-controlled workflows. In *CODASPY*. http://dx.doi.org/10.1145/2857705.2857716.

Crampton, J., Huth, M., and Kuo, J. H.-P. (2014). Authorized workflow schemas: deciding realizability through LTL model checking. *Int J Softw Tools Technol Transfer*, 16(1):31–48, http://dx.doi.org/10.1007/s10009-012-0269-3.

Hunsberger, L., Posenato, R., and Combi, C. (2012). The Dynamic Controllability of Conditional STNs with Uncertainty. In *PlanEx at ICAPS 2012*, pages 1–8. http://arxiv.org/abs/1212.2005.

Hunsberger, L., Posenato, R., and Combi, C. (2015). A sound-and-complete propagation-based algorithm for checking the dynamic consistency of conditional simple temporal networks. In *TIME 2015*, pages 4–18. http://dx.doi.org/10.1109/TIME.2015.26.

Lenz, R. and Reichert, M. (2007). It support for healthcare processes - premises, challenges, perspectives. *Data Knowl. Eng.*, 61(1):39–58, http://dx.doi.org/10.1016/j.datak.2006.04.007.

Morris, P. H., Muscettola, N., and Vidal, T. (2001). Dynamic control of plans with temporal uncertainty. In *IJCAI 2001*, pages 494–502.

Wang, Q. and Li, N. (2010). Satisfiability and resiliency in workflow authorization systems. *ACM Trans. Inf. Syst. Secur.*, 13(4).

APPENDIX

Theorem 1. Encoding ACTNs into TGAs has polynomial-time complexity.

Proof. The main components having a role in the complexity analysis of the encoding of an ACTN are:

(i) time points, (ii) authorized users for each control time point, (iii) authorization constraints, and (iv) different labels in the ACTN.

For each control time point, there is a self-loop transition for any authorized user. These transitions contain in their guard the scenario in which the time point has to be executed and may contain additional conditions about the predecessors. From § 4, the complexity of finding the predecessors for a time point X is linear in the number of all authorization constraints. Thus, the generation of these transitions is a polynomial-time task.

Transitions modeling the execution of the contingent time points and assigning truth values are the same as those given for CSTNUs (we just extend the update part), and they are not even replicated for the authorized users. Fail transitions and the transitions regulating the game interplay remain exactly the same as well. Thus, the time complexity of the generation of this set of transitions remains polynomial.

We are left to prove that the generation of the winning path is a polynomial-time task too. The main problem is that such a path consists of intermediate locations that must allow one to consider all possible combinations of the disjunctions expressed as authorization constraints. The generation of the winning path is the same of that for CSTNUs (augmented with the checks for the authorization policies) when we consider the set of core constraints (polynomial). When considering the non core authorization constraints we create as many intermediate locations as the number of disjunctions. However, we remark that our encoding does not generate any combination of the `sat` transitions nor a combination of the locations. The model-checking phase will take care of exploring them all. Thus, the generation of the winning path has linear complexity in the number of the different labels with respect to the number of the possible disjunctions.

Since all operations have polynomial complexity, the overall complexity is polynomial as well. □

Theorem 2. Encoding ACTNs into TGAs correctly captures the execution semantics given in § 4.

Proof. We show that any sequence of partial schedules that can be generated for any ACTN according to the execution semantics given in § 4 corresponds to a run for the equivalent TGA that can be generated by following its transitions according to the classic TGA semantics. We extend the proof of correctness given in (Cimatti *et al.*, 2016) to accommodate users and authorization constraints. The proof is by induction.

Each respectful partial schedule that can be generated for the ACTN corresponds to a state of the TGA in which: the location is `env`, $c_\delta = 0$, $last(OccTP) = \hat{c}$, and for each executed time point X, $time(X) = \hat{c} - clk[X]$, $xtd[X] = \top$, $user(X) = SysTrace[X]$, whereas $time(X) = \hat{c}$, $xtd[X] = \bot$ and $user(X)$ if X hasn't been executed yet. If $P?$ is an observation time point, then *KnownProp* contains (p,b) for $b \in \{\top,\bot\}$ if $P?$ has been executed, and doesn't contain it otherwise. Also, if some activation point has been executed by the user u and the related contingent has not, then $Avail[u] = \bot$, else $Avail[u] = \top$. Note that $Avail[u] = \top$ and $Avail[u] = \bot$ (in the TGA) mean $u \in Avail$ and $u \notin Avail$ in the execution semantics given in § 4 (where *Avail* is a set).

Base Case. The initial $\mathcal{PS}$ corresponds to the initial state of the TGA in which the location is `env`, $\hat{c} = 0$ all clocks $clk[X] = 0$, all $xtd[X] = \bot$, all $SysTrace[X] = -1$, all $prop[p] = 0$, and $Avail[u] = \top$. This partial schedule is trivially respectful.

Inductive Step. Suppose that $\mathcal{PS}$ is a respectful partial schedule that can be generated according to the execution semantics for ACTNs, and that $\mathcal{PS}$ satisfies the invariant that we required of each respectful partial schedule at the beginning of this proof.

Let θ be the corresponding state of the TGA. Since $c_\delta = 0$, the only transitions that are immediately enabled are those handling contingent time point executions and truth value assignments. These transitions, if taken, correspond to `env`'s instantaneous reactions, in which a set of one or more contingent time-points can be executed simultaneously or some propositions can be assigned a value. Suppose that `env` doesn't take any transition when $c_\delta = 0$. As soon as $c_\delta > 0$, both `ctrl` and `env` have transitions that they could take at any time with respect to the enforced condition over the predecessors. For example, `env` might decide to execute one or more contingent time-points $C_1,\ldots,C_n$ when $c_\delta = 3$. That would correspond to $\Delta_{\texttt{env}} = (k,\{C_1,\ldots,C_n\})$, where $k = last(OccTP)+3$.

Since each time `env` takes a transition c_δ is reset to 0, `ctrl` is unable to interrupt `env` while `env` is executing contingent time points and assigning truth values to propositions. Thus, at those time instants $\Delta_{\texttt{ctrl}} = \texttt{wait}$ and the resulting outcomes are exactly the cases 1-2 of Definition 7. The guard of `env`'s transition, enforcing the duration bounds for a contingent link (A,x,y,C), ensures that the resulting partial schedule is respectful as C can only be executed such that $C - A \in [x,y]$ (analogous to predecessors). Likewise, for a truth value assignment the fail transition that `ctrl` can take (if $\delta > 0$) ensures that `env` can assign a truth value to a proposition instantaneously after the execution of the observation time point (otherwise, `ctrl` could trivially move to `goal`).

Also, when `env`'s sequence of "simultaneous"

transitions completes, $\hat{c}$ equals the time of the most recent execution (e.g., $last(OccTP)+3$). In addition, for each newly executed time-point C, the clock $clk[C]$ is reset $xtd[C]$ is set to $\top$, $SysTrace[C]$ is set to $SysTrace[A]$ and $Avail[SysTrace[C]] = \top$. Since $clk[C]$ is reset only once, $\hat{c} - clk[C]$ remains fixed forever.

Instead, suppose that `ctrl` has decided to commit a set of users to execute a set of control time points before `env` executes his, say at time $last(OccTP)+2$. This situation results in `ctrl` taking the `gain` transition to take back control and then, once in its location, instantaneously commit the users to execute the time points at that time, blocking all users executing an activation time point, and immediately returning to the `env` location by means of the `pass` transition. Since the location of `ctrl` is urgent, $\hat{c} = last(OccTP)+2$ when the `pass` transition is taken. This sequence of transitions corresponds to the partial outcome in Definition 7 (cases 3-4) where $\Delta_{\texttt{ctrl}} = (t, \{(u_1, X_1), \ldots, (u_n, X_n)\})$, $t = last(OccTP)+2$, and for each $(u, X) \in ControlTP$ (of $\Delta_{\texttt{ctrl}}$), $u \in \mathcal{A}(X)$. Moreover, if `env` chooses to instantaneously execute some contingent time point at the same time $last(OccTP)+2$, that will correspond to an instantaneous reaction.

Finally, if at time $last(OccTP)$, `ctrl` and `env` both decide to execute some time points at time $last(OccTP)+1$, then the ACTN semantics (inheriting the CSTNU semantics) ensures that `ctrl`'s time points are executed first, and that `env` is able to instantaneously react if it decides to do so (equivalent to `ctrl`'s transitions having priority over `env`'s). As soon as the execution returns to the location of `env`, the $\hat{c}$ will still be $last(OccTP)+1$ (because, again, time has not elapsed at `ctrl`). Since, in all cases, the resulting state of the TGA satisfies the desired invariant property, the result is proven. □

Theorem 3. Let $\mathcal{S}$ be any ACTN, $\mathcal{G}$ be the encoding of $\mathcal{S}$, and $\sigma_{\mathcal{G}}$ be a winning TGA counter-strategy for `ctrl`. Then there is an equivalent RTED-based strategy $\sigma_{\texttt{ctrl}}$ for `ctrl` that will ensure the satisfaction of all authorization constraints in $\mathcal{S}$ whatever the contingent durations and truth value assignments.

Proof. If $\mathcal{S}$, $\mathcal{G}$, $\sigma_{\mathcal{G}}$ are as assumed, then $\sigma_{\mathcal{G}}$: $Loc \times V \to Act \cup \texttt{wait}$, with Act the set of `ctrl`'s actions (equivalently the set of uncontrollable transitions) and V abstracts the internal state of the TGA.

Suppose the TGA has just got into the state $(\texttt{env}, V)$. As we have already noted, for any time point X, associated clock $clk[X]$ and Boolean variable $xtd[X]$, we have: $xtd[X] = \bot$, $clk[X] = \hat{c}$, and $SysTrace[X] = -1$ before X executes, and $xtd[X] = \top$, $clk[X] < \hat{c}$, and $SysTrace[X] = u$ after X executed. For each observation time point $P?$, the associated proposition modeled by $prop[p]$ is 0 (i.e., unknown) before $P?$ executes, and either 1 (i.e., $\top$) or -1 (i.e., $\bot$) after $P?$ executed. Thus, V specifies a partial schedule.

Now, suppose that $last(OccTP) < \hat{c}$ (i.e., that some positive time has elapsed since the last execution event in $\mathcal{PS}$). If nothing has happened, it means that there has been a sequence of `gain` and `pass` transitions going back and forth between `env`'s and `ctrl`'s locations. In such a loop, `ctrl` has not executed any control point, and `env` has just waited. Let $(\texttt{env}, V')$ be the state immediately preceding such loop. Then, for some positive $\varepsilon > 0$, all the clocks in V equal those in V' plus ε, and by construction, $last(OccTP)$ refers to the clocks in V'. We abuse notation and write $V + k$ meaning that all values of the clocks in V are augmented by k. Next, let $d = \min\{d \mid \sigma_{\mathcal{G}}(\texttt{env}, V'+d) \neq \texttt{wait} \wedge \sigma_{\mathcal{G}}(\texttt{env}, V'+d) \neq \texttt{pass}\}$ be the minimum time that can elapse from V before the strategy $\sigma_{\mathcal{G}}$ recommends a transition different from `gain` and `pass`, and let $V_0 = V_0 + d$. The unique sequence of execution transitions at `ctrl` is $\tau_1 = \sigma_{\mathcal{G}}(\texttt{ctrl}, V_0), \ldots, \tau_n = \sigma_{\mathcal{G}}(\texttt{ctrl}, V_n)$, where each $V_{i+1} = V_i$, except for the $clk[X]$ with X the time point executed by τ_i. The termination of this sequence of transitions is guaranteed since time points are finite and can only be executed once. If τ_n is the last execution transition, then $\texttt{pass} = \sigma_{\mathcal{G}}(\texttt{ctrl}, V_n)$. That transition leads back to the state (env, V_n), where V_n is the same as V_0, except that the clocks for the time points executed by the transitions, $\tau_1, \ldots, \tau_n$, are all 0 in V_n.

Next, let t be the time at which $\sigma_{\mathcal{G}}$ recommends `ctrl` a non-trivial transition, and $ControlTP$ be the set of time-points corresponding to the execution transitions, $\tau_1, \ldots, \tau_n$. Then $(t, ControlTP)$ is a $\Delta_{\texttt{ctrl}}$ corresponding to what the strategy recommends at $(\texttt{env}, V_0)$. Note that `env` may decide to instantaneously react by executing some contingent points at time t too, an outcome that is prevented by the execution semantics for ACTNs (Definition 7, cases 3-4). Finally, `env` may decide to intervene before time t arrives, by executing one or more contingent time-points and effectively generating a new partial schedule $\mathcal{PS}'$. In that case, the same procedure could be applied to $\mathcal{PS}'$ to generate an appropriate $\Delta_{\texttt{ctrl}}$. Since the guard on the transition from `env` to `ctrl` requires a positive time delay, that $\Delta_{\texttt{ctrl}}$ is properly prohibited from any kind of instantaneous reaction (by `ctrl`). This procedure gives a mapping from any $(\texttt{env}, V)$ state that is reachable following $\sigma_{\mathcal{G}}$. The sequences of partial schedules generated by following the RTEDs correspond to runs that can be produced by $\sigma_{\mathcal{G}}$. Thus, the complete schedules generated by the RTEDs satisfy all authorization constraints. □