Journal of Artificial Intelligence Research 64 (2019) 931–985 Submitted 07/2018; published 04/2019

Conditional Simple Temporal Networks with Uncertainty and Resources

Carlo Combi CARLO.COMBI@UNIVR.IT
Dipartimento di Informatica, Università di Verona
strada le grazie 15, 37134 Verona, Italy

Roberto Posenato ROBERTO.POSENATO@UNIVR.IT
Dipartimento di Informatica, Università di Verona
strada le grazie 15, 37134 Verona, Italy

Luca Viganò LUCA.VIGANO@KCL.AC.UK
Department of Informatics, King's College London
30 Aldwych, London E7 9QU, United Kingdom

Matteo Zavatteri MATTEO.ZAVATTERI@UNIVR.IT
Dipartimento di Informatica, Università di Verona
strada le grazie 15, 37134 Verona, Italy

Abstract

Conditional simple temporal networks with uncertainty (CSTNUs) allow for the representation of temporal plans subject to both conditional constraints and uncertain durations. Dynamic controllability (DC) of CSTNUs ensures the existence of an execution strategy able to execute the network in real time (i.e., scheduling the time points under control) depending on how these two uncontrollable parts behave. However, CSTNUs do not deal with resources.

In this paper, we define conditional simple temporal networks with uncertainty and resources (CSTNURs) by injecting resources and runtime resource constraints (RRCs) into the specification. Resources are mandatory for executing the time points and their availability is represented through temporal expressions, whereas RRCs restrict resource availability by further temporal constraints among resources.

We provide a fully-automated encoding to translate any CSTNUR into an equivalent timed game automaton in polynomial time for a sound and complete DC-checking.

1. Introduction

Temporal aspects have been studied both for business processes and for planning (Kafeza & Karlapalem, 2000; Eder, Gruber, & Panagos, 2000; Chinn & Madey, 2000; Bettini, Wang, & Jajodia, 2002; Combi, Gambini, Migliorini, & Posenato, 2014a; Combi & Posenato, 2009, 2010; Cimatti, Do, Micheli, Roveri, & Smith, 2018). The most commonly considered temporal aspects are temporal constraints on task durations, deadlines, inter-task temporal synchronizations, temporal uncertainty for task durations, and uncontrollable conditional constraints, i.e., constraints that must be considered according to some external conditions. As an example of temporal uncertainty for task duration (also known as *contingent duration*), consider the writing activity of a paper to be submitted to a conference. Once it has started, it will last at least a minimum amount of time to allow the authors to get to a polished version and at most a time determined by the deadline of the conference.

However, the exact moment when the authors will have it finished is unknown at design time. Moreover, after the submission, in case of acceptance of the paper, further temporal constraints will be considered for preparing the camera-ready copy. In case of rejection, no further constraints have to be considered. Such a situation represents a case of uncontrollable conditional constraints as the decision of acceptance/rejection is not under the control of the system and it can be known only at runtime.

A further temporal aspect for business processes and for planning is related to the scheduling of resources for executing a process (Avanes & Freytag, 2008; Watahiki, Ishikawa, & Hiraishi, 2011; Qi, Wang, Muñoz-Avila, Zhao, & Wang, 2017). Resources may represent both human and device-based agents, responsible for the execution of some task. For example, an aircraft pilot must rest at least 8 hours before he is allowed to pilot again (i.e., before being again available). Similarly, a surgeon who has just carried out a 4 hour intervention must rest at least 2 hours before starting another one.

All such aspects cannot be considered in isolation as they are intertwined with other constraints in real contexts. For example, the constraint related to the pilot rest has to be merged with other constraints related to flight schedules/durations.

In this paper we propose a new model for representing processes with both temporal constraints, possibly having different kinds of uncertainty, and resource constraints having temporal features; we call this model Conditional Simple Temporal Network with Uncertainty and Resources (CSTNUR). Moreover, we propose a technique for deriving at design time whether such a process can be executed satisfying all the given constraints.

The CSTNUR model adopts a network-based representation of processes. Temporal networks have long been studied for the modeling, validation and execution of process plans subject to temporal constraints. The main components of a temporal network are *time points* (e.g., variables having continuous domain) modeling the occurrence of events, and *temporal constraints*, usually related to minimal and maximal temporal distances between pairs of time points. An execution of a temporal network consists of assigning real values to the time points (i.e., a scheduling), such that all temporal constraints are satisfied.

1.1 Contributions

The novelty of our proposal is three-fold:

1. CSTNURs allow us to represent:
   (a) both time points that are under the control of the system and time points that can be only observed when they happen (contingent time points);
   (b) execution scenarios that are not under the control of the system;
   (c) resource assignment constraints possibly depending on some temporal constraints.

   Such constraints specify whether a resource can be committed to execute a time point and may involve the specification of the execution times of (other) time points and, thus, need to be evaluated at runtime. The temporal processes we consider in this paper specify authorized resources, in charge of executing the time points, and *runtime resource constraints (RRCs)* saying *when* and *which* resources are available to execute a task according to when and which other resources have executed some other tasks in

the past. In other words, resources are assigned a further temporal expression whose evaluation with respect to the global time models their availability, whereas RRCs operate on these temporal expressions. As a result, resources may be available in some time intervals and may turn unavailable in some other intervals (or vice versa).

2. We formalize *Dynamic Controllability* (DC) for CSTNURs. Dynamic controllability means that a CSTNUR can be executed by suitable resources, satisfying all the given constraints no matter what are the values of the uncontrollable parts (i.e., contingent time points and conditions). We give this formalization in terms of *real time execution decisions (RTEDs)*, already used in Hunsberger (2009), Cimatti, Hunsberger, Micheli, and Roveri (2014), Cimatti, Hunsberger, Micheli, Posenato, and Roveri (2014, 2016), and extended here to consider resource assignments and related constraints.

3. DC can be checked at design time. To verify whether a CSTNUR is DC, we propose an encoding of CSTNURs into Time Game Automata (TGA) and show how dynamic controllability checking corresponds to winning a two-player game. Such an encoding improves and extends the encoding proposed by Cimatti et al. (2016) to consider resources and RRCs. We formally prove the correctness of the proposed encoding and discuss its complexity.

1.2 Organization

The rest of the paper is organized as follows. Section 2 provides a motivating example we use throughout the paper. Section 3 provides background on CSTNUs and their sound and complete DC-checking via TGAs. Section 4 defines CSTNURs along with their execution semantics in terms of RTEDs. Section 5 extends the encoding given in Section 3 by taking into consideration resources and RRCs. Section 6 discusses the complexity and correctness of the encoding. Section 7 discusses related work. Section 8 sums up our proposal. In Appendix A, we propose the modeling and validation of our motivating example by using Uppaal-TIGA.

2. Motivating Example

As a motivating example, we consider (a simplification of) a temporal workflow modeling a round-trip flight from Anchorage, Alaska to Frankfurt, Germany (direct flights). We show its graphical workflow-representation in Figure 1 through an extended BPMN notation (Posenato et al., 2018). Such a notation has been completed by adding available resources for tasks ($\{\texttt{Alice}, \texttt{Bob}\}$ are available for executing task `Deicing`). We focus on the part involving pilots and engineers. Once boarding is complete, take-off could be delayed due to extreme weather conditions and related safety procedures such as, for example, deicing. Deicing is the process of removing snow and ice from the plane surfaces (especially wings) by "power washing" the aircraft with chemicals which will also remain on the surfaces in order to prevent the reformation of the ice. This (uncontrollable) condition is modeled by a conditional split connector (diamond labeled by `Deicing?`). If $\texttt{Deicing?} = \top$ (i.e., `Deicing?` is true) then the `Deicing` process starts after minimum 5 minutes and within 10

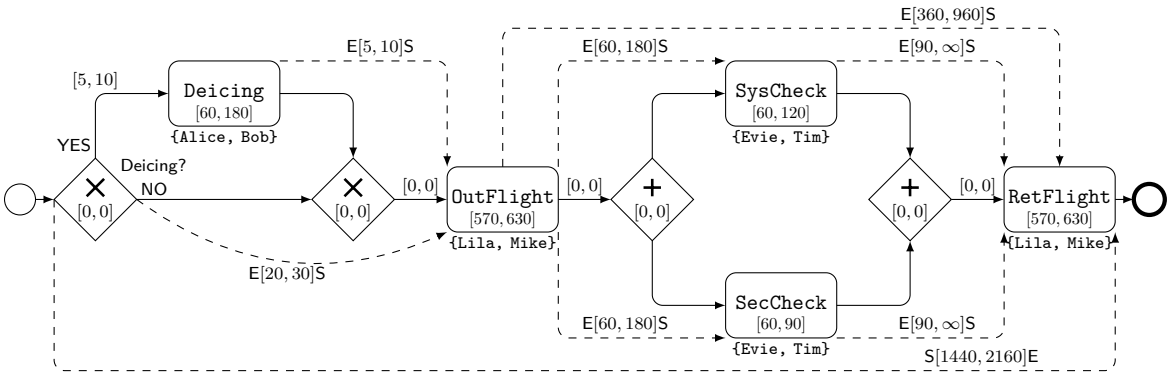


Figure 1: Example of a temporal workflow for a round-trip flight in extended BPMN (Posenato et al., 2018) considering also resources available for the execution of tasks ($\{\texttt{Alice}, \texttt{Bob}\}$ are available for executing task `Deicing`). All ranges are in minutes. Missing ranges have value $[0, \infty]$. Dashed edges represent inter-task temporal constraints.

minutes (`Yes` branch). This task lasts from 1 to 3 hours[1]. After `Deicing` has finished, the plane takes off after minimum 5 and within 10 minutes. `Alice` and `Bob` are two specialized workers who can be committed for this task. Instead, if `Deicing?` $= \bot$ (i.e., `Deicing?` is false), the plane just takes off after 20 and within 30 minutes modeling the time needed to provide passengers with the safety instructions and reach the runway (`No` branch). Note that in case of deicing, there is plenty of time for the safety instructions. Once the aircraft has taken off, the outbound flight (`OutFlight`) lasts from 9 hours and 30 minutes to 10 hours and 30 minutes. `Lila` and `Mike` are pilots who can be committed for this task. Once the aircraft has landed, a system check (`SysCheck`) and a security check (`SecCheck`) start after minimum 1 and within 3 hours. `Evie` and `Tim` are two engineers who can be committed for these two tasks. `SysCheck` lasts 1 to 2 hours, whereas `SecCheck` lasts 1 to 1 hour and a half. Once both these two tasks are done, the plane can take off again after minimum 6 and maximum 16 hours since its landing (`RetFlight`) with the same pilots available for this task. The whole process lasts minimum 24 and maximum 36 hours.

This process employs users as resources and enforces two safety properties. First, the process enforces the USA Federal Aviation Regulations for flight time limitations and rest requirements (FAR, 2019) saying that after a 10–12 hour (multi-time zone) flight, a pilot must rest from 14 to 18 hours before piloting again (resource constraint). Second, we require that if `SysCheck` and `SecCheck` are executed in parallel, they are not executed by the same engineer (who can however execute both sequentially). For the sake of simplification, in this paper we assume that one resource only is committed for executing each task (equivalently, each task is executed by a single user).

According to this scenario, temporal constraints are intertwined with resource constraints and, therefore, it is necessary to verify whether it is possible to determine a schedule

1. Actually, deicing an aircraft does not take 3 hours, but since all leaving aircrafts have to do so following the departure scheduling, each plane queues for its turn.

for executing such a process satisfying all temporal and resource constraints. Such a schedule must be dynamic, i.e., it must assign the starting time of different tasks together with resource assignment knowing only the durations and resource assignment of past tasks. In this way, the schedule could be more flexible with respect to other kind of schedules, i.e., a schedule that requires to know in advance all task durations. Indeed, task durations are not under the control of the schedule and they are known only once the tasks are executed.

Thus, we need to represent both temporal constraints and resource ones in a single formalism and to verify some global properties.

3. Background

A *Simple Temporal Network (STN) (Dechter, Meiri, & Pearl, 1991)* is able to model a temporal plan in which it is possible to constrain the distance between pairs of time points and the occurrence of all time points is under the control of the executing agent (i.e., a real-time planner). For each pair of time points, the temporal distance can be limited to stay in a range of real values. Consistency analysis is able to determine whether it is possible to schedule time points such that all given temporal constraints are not violated. The decision problem of consistency for STNs has polynomial-time complexity (i.e., it is in the P class) (Dechter et al., 1991).

One of the most important extensions of STN is the *Simple Temporal Network with Uncertainty (STNU)*. It extends an STN by adding contingent links to model uncontrollable (but bounded) durations (Vidal & Ghallab, 1996; Vidal & Fargier, 1997, 1999; Morris, Muscettola, & Vidal, 2001). The controller executing an STNU can only decide when to execute the activation time point of a contingent link, but it merely observes the occurrence of the corresponding contingent time point. The dynamic controllability checking aims to verify the existence of an execution strategy to execute the time points under control of the system such that all constraints are satisfied for any possible (combination of) durations of contingent links. The decision problem of dynamic controllability for STNUs is in P (Morris & Muscettola, 2005).

Conditional Simple Temporal Networks (CSTNs) extend STNs by introducing conditions that represent different execution scenarios. Each time point/constraint may have a label (conjunction of condition variables) that represents in which scenario it has to be executed/ satisfied.

Conditional Simple Temporal Networks with Uncertainty (CSTNUs) merge the semantics of STNUs and CSTNs in order to deal with uncertain durations and conditional constraints simultaneously.

In the following section, we provide more details on CSTNUs. We begin by giving a few useful definitions for label, label entailment and label consistency. Then, we summarize the formal specification of CSTNUs and the related notion of dynamic controllability. Finally, we present a recent encoding of CSTNUs into Timed Game Automata (TGAs) for a sound-and-complete dynamic controllability checking (DC-checking).

3.1 Conditional Simple Temporal Networks with Uncertainty (CSTNUs) and the Dynamic Controllability Checking

Given a set $\mathcal{P}$ of propositional letters, a *label* ℓ is any 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 labels whose literals are drawn 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 $\ell_1 \Rightarrow \ell_2$) if and only if all literals in ℓ_2 appear in ℓ_1 too (i.e., if ℓ_1 is more *specific* than ℓ_2). For instance, if $\ell_1 = p \wedge \neg q$ and $\ell_2 = p$, then ℓ_1 and ℓ_2 are consistent since $p \wedge \neg q \wedge p$ is satisfiable, and ℓ_1 entails ℓ_2 since $p \wedge \neg q \Rightarrow p$. The vice versa does not hold in general (e.g., $p \not\Rightarrow p \wedge \neg q$).

The following definition summarizes the definition of CSTNUs (Hunsberger, Posenato, & Combi, 2012; Combi, Hunsberger, & Posenato, 2014b) and some basic properties presented by Hunsberger, Posenato, and Combi (2015) that must hold in such networks.

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

(1) $\mathcal{T} = \{X, Y, \dots\}$ is a finite set of *time points* (i.e., variables with continuous domain).

(2) $\mathcal{P} = \{p, q, \dots\}$ 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 $X \in \mathcal{T}$.

(4) $\mathcal{OT} \subseteq \mathcal{T}$ is a set of *observation time points*.

(5) $O \colon \mathcal{P} \to \mathcal{OT}$ is a bijection associating a unique observation time point to each propositional letter.

(6) $\mathcal{C}$ is a set of *labeled constraints* each one having the form $(l \leq Y - X \leq u, \ell)$, where $X, Y \in \mathcal{T}$, $l, u \in \mathbb{R}$ with $l \leq u$ and $\ell \in \mathcal{P}^*$. If, given $X, Y \in \mathcal{T}$ and $\ell \in \mathcal{P}^*$, there exist two constraints $(l_1 \leq Y - X \leq u_1, \ell)$ and $(l_2 \leq Y - X \leq u_2, \ell)$ in $\mathcal{C}$, then $l_1 = l_2$ and $u_1 = u_2$, i.e., the time distance range between X and Y is unique with respect to ℓ.

(7) $\mathcal{L}$ is a set of *contingent links* each having the form (A, x, y, C), where $A, C \in \mathcal{T}$ are different time points (written $A \not\equiv C$), $0 < x < y < \infty$ and $L(A) = L(C)$. For any pair (A_1, x_1, y_1, C_1), $(A_2, x_2, y_2, C_2) \in \mathcal{L}$ with $A_1 \not\equiv A_2$ we have that $C_1 \not\equiv C_2$.

(8) For each constraint $(l \leq Y - X \leq u, \ell) \in \mathcal{C}$, we have that $\ell \Rightarrow L(Y) \wedge L(X)$. This property is called *constraint label coherence* (Hunsberger et al., 2015).

(9) For each literal p or $\neg p$ appearing in ℓ, we have that $\ell \Rightarrow L(O(p))$. This property is called *constraint label honesty* (Hunsberger et al., 2015).

(10) For each $X \in \mathcal{T}$, if literal p or $\neg p$ appears in $L(X)$, then $O(p)$ has to occur before X, i.e., $(\epsilon \leq X - O(p) \leq +\infty, L(X)) \in \mathcal{C}$ for some $\epsilon > 0$, where ϵ is a real constant modeling a non-instantaneous reaction time. This property is called *time point label honesty* (Hunsberger et al., 2015).

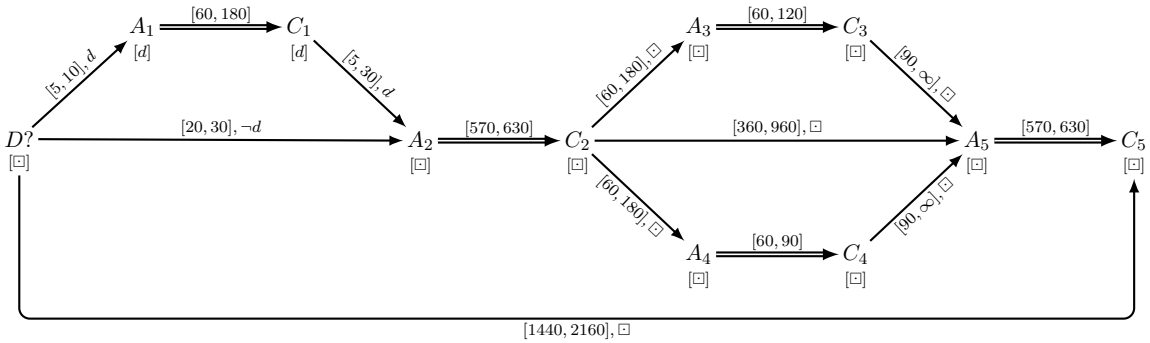


Figure 2: CSTNU modeling the temporal plan in Figure 1 without any resource specification. $D?$ models the conditional split connector, whereas $A_1 \Rightarrow C_1$, $A_2 \Rightarrow C_2$, $A_3 \Rightarrow C_3$, $A_4 \Rightarrow C_4$ and $A_5 \Rightarrow C_5$ model tasks `Deicing`, `OutFlight`, `SysCheck`, `SecCheck` and `RetFlight`.

In a contingent link (A, x, y, C), A is called *activation time point*, whereas C is called *contingent time point*. Once A is executed, C is only observed to occur. However, C is guaranteed to occur after A, satisfying the constraint $C - A \in [x, y]$. Moreover, a contingent link has an implicit label given by the label $\ell = L(A) = L(C)$.

In general, a non-contingent time point is *executed* by the system executing the network by assigning it a real value when its label is true. Instead, for each contingent link, once its activation time point has been executed, the corresponding contingent time point is executed by the *environment*, which assigns it a real value among those allowed by the range of the corresponding contingent link. If a time point is also an observation one, then the value of associated proposition is set by the environment when it is executed.

The truth values of the propositions and the durations of contingent links in a CSTNU are not known in advance. Instead, they are incrementally revealed over time as the corresponding observation and contingent time points are executed, respectively. A *dynamic execution strategy* executes the time points in a CSTNU, by reacting to the truth value assignments and to the occurrence of the contingent time points in real time.

A *viable* and *dynamic execution strategy* for a CSTNU is a strategy guaranteeing that all relevant constraints, i.e., constraints having true label, will be satisfied no matter which truth values for the propositions and which durations for the contingent links are incrementally revealed over time. We say that a CSTNU having such a strategy is *dynamically controllable*.

Figure 2 depicts a graphical representation of a CSTNU modeling the temporal plan in Figure 1 without any resource specification[2]. Nodes represent time points, single arrows represent labeled constraints, whereas double ones contingent links. Each node/single arrow is labeled by a label ℓ specifying in which scenario such a component must be considered. $D?$ models the conditional split connector (diamond), whereas $(A_1, 60, 180, C_1)$, $(A_2, 570, 630, C_2)$, $(A_3, 60, 120, C_3)$, $(A_4, 60, 90, C_4)$ and $(A_5, 570, 630, C_5)$ model `Deicing`,

2. This representation is a simplified one obtained considering the mapping presented by Posenato et al. (2018) where nodes that have $[0, 0]$ distance have been collapsed in a single CSTNU node.

`OutFlight`, `SysCheck`, `SecCheck` and `RetFlight`. The execution of the observation time point $D?$ entails a truth value assignment to the associated proposition d.

If d is assigned $\top$, the contingent link $(A_1, 60, 180, C_1)$ is considered, else ignored. Then, all other activation time points A_2, A_3, A_4 and A_5 are executed such that all constraints are satisfied whatever the real value assignments to the corresponding C_2, C_3, C_4 and C_5.

In order to formally characterize the execution semantics of CSTNUs, we introduce the notion of *real-time execution decisions (RTEDs)* and *instantaneous reactions*. We model the execution of a CSTNU as a two-player game: the *controller* `ctrl` and the *environment* `env` (Cimatti et al., 2016). Intuitively, an RTED is a decision to execute a set of time points or a decision to wait for something to happen (i.e., waiting for the occurrence of some contingent time points), whereas an instantaneous reaction is the capability of the environment to react to the controller's actions by executing, instantaneously, some contingent time points or by setting the truth values of propositions associated to the (possible) just executed observation time points.

For a CSTNU it is necessary to specify RTEDs for both `ctrl` and `env`, and instantaneous reactions for `env` only. `ctrl`'s RTEDs model a strategy for executing all relevant non-contingent time points of the network, whereas `env`'s RTEDs model an unknown strategy for executing contingent time points. Likewise, the instantaneous reactions represent an unknown strategy for both reacting to `ctrl`'s actions and setting the truth values of the propositions.

The purpose of `ctrl` is that of executing all relevant non-contingent time points such that constraints involving those time points will eventually be satisfied no matter what durations for contingent links and truth values for propositions the environment decides.

For example, suppose that `env`'s RTED (in words) is "if nothing happens before or *at time* 10, then `env` shall execute C at time 10", and suppose that at 10, `ctrl` executes B. `env` can withdraw its decision to execute C and instantaneously react by executing some other contingent time point(s) at time 10.

We now introduce the notion of *partial schedule*, which we will use to define the RTEDs and the instantaneous reactions.

Definition 2 (Partial Schedule)**.** A *partial schedule* for a CSTNU is a pair $\mathcal{PS} = (\mathit{Executed}, \mathit{Assigned})$, where

- *Executed* is a set of pairs (X, k) meaning that time point X was executed at time k. We shorten the set of time points and execution times of the elements in this set as $\mathit{Executed}_{\mathcal{T}} = \{X \mid (X, k) \in \mathit{Executed}\}$ and $\mathit{Executed}_{\mathbb{R}} = \{k \mid (X, k) \in \mathit{Executed}\}$, respectively. If $(X, k) \in \mathit{Executed}$, then $\mathit{time}(X) = k$. $\mathit{last}(\mathit{Executed}) = \max\{v \mid v \in \mathit{Executed}_{\mathbb{R}}\}$ represents the time instant of the last executed time point; when $\mathit{Executed} = \emptyset$, we assume $\mathit{last}(\mathit{Executed}) = -\infty$.

- *Assigned* is a set of pairs (p, b) meaning that proposition p was assigned $b \in \{\top, \bot\}$. $\ell_{cps} = \{p \mid (p, \top) \in \mathit{Assigned}\} \cup \{\neg q \mid (q, \bot) \in \mathit{Assigned}\}$ represents the current partial scenario according to $\mathcal{PS}$.

A partial schedule is called *locally consistent* (formerly, *respectful* (Cimatti et al., 2014)) if all constraints $(x \leq Y - X \leq y, \ell)$ of the considered CSTNU such that $\ell_{cps} \Rightarrow \ell$ and

$Y, X \in \mathit{Executed}_{\mathcal{T}}$ are satisfied. Finally, $\mathcal{PS}^*$ represents the set of all possible partial schedules.

We can now proceed by defining `ctrl`'s and `env`'s RTEDs.

Definition 3 (RTED for `ctrl`)**.** An *RTED for* `ctrl`, in symbols, $\Delta_{\texttt{ctrl}}$, specifies which action has to be performed by `ctrl` during an execution (represented by a considered $\mathcal{PS}$). It has two forms: `wait` or $(t, \mathit{NonContingent})$.

- $\Delta_{\texttt{ctrl}} = \texttt{wait}$ means that `ctrl` decides to do nothing and waits for some contingent time point to occur. It is applicable only if at least one contingent link has been activated (i.e., the activation time point A has been executed but the related contingent C has not).
- $\Delta_{\texttt{ctrl}} = (t, \mathit{NonContingent})$ represents the conditional constraint: "if `env` does nothing *before* time t, then `ctrl` shall execute the time points in *NonContingent* at time t". Such an RTED is applicable if and only if $t > \mathit{last}(\mathit{Executed})$, *NonContingent* is a non-empty set of unexecuted non-contingent time points (i.e., $\mathit{NonContingent} \subseteq \mathcal{T} \setminus \mathit{Executed}_{\mathcal{T}}$ and $\mathit{NonContingent} \neq \emptyset$) and $\ell_{cps} \Rightarrow L(X)$ for each $X \in \mathit{NonContingent}$ (i.e., X must be considered in the current partial scenario).

The set of all RTEDs for `ctrl` is denoted by $\Delta^*_{\texttt{ctrl}}$.

Definition 4 (RTED for `env`)**.** An *RTED for* `env`, in symbols, $\Delta_{\texttt{env}}$, specifies which action has to be performed by `env` during an execution. It has two forms: `wait` or $(t, \mathit{Contingent})$.

- $\Delta_{\texttt{env}} = \texttt{wait}$ means that `env` decides to do nothing. It is applicable only if no contingent link has been *activated*, i.e., its activation time point has been already executed.
- $\Delta_{\texttt{env}} = (t, \mathit{Contingent})$ represents the conditional constraint: "if `ctrl` does nothing *before or at time* t, then `env` shall execute the contingent time points in *Contingent* at time t". Such a decision is applicable if and only if $t > \mathit{last}(\mathit{Executed})$, *Contingent* is a non-empty subset of unexecuted contingent time points. That is, $\mathit{Contingent} = \{C \mid (A, x, y, C) \in \mathcal{L} \wedge A \in \mathit{Executed}_{\mathcal{T}} \wedge t \in [\mathit{time}(A) + x, \mathit{time}(A) + y]\}$.

The set of all RTEDs for `env` is denoted by $\Delta^*_{\texttt{env}}$.

We now consider the instantaneous reactions for `env`. Such reactions entail that `env` operates instantaneously on some uncontrollable part(s). For example, `env` could execute some, not previously planned, contingent time points or assign truth values to the propositions associated to a set of just executed observation time points, or do both actions. Of course, more instantaneous reactions are possible at the same time.

Definition 5 (Instantaneous Reaction for `env`)**.** Let $\mathcal{PS} = (\mathit{Executed}, \mathit{Assigned})$ be a partial schedule. Let χ^0 be the set of contingent time points C_i such that $A_i \in \mathit{Executed}_{\mathcal{T}}$ and $\mathit{time}(A_i) + y_i = \mathit{last}(\mathit{Executed})$, i.e., their execution windows terminate precisely at $\mathit{last}(\mathit{Executed})$. Let $\chi^\star$ be any (possibly empty) subset of the contingent time points C_i such that $A_i \in \mathit{Executed}_{\mathcal{T}}$ and $\mathit{last}(\mathit{Executed}) \leq \mathit{time}(A_i) + y_i$, i.e., their execution windows include $\mathit{last}(\mathit{Executed})$. Finally, Let $\mathcal{I}_C = \chi^0 \cup \chi^\star$. An *instantaneous reaction* $\mathcal{IR}$ for `env` is a decision

(1) to execute all contingent time points in $\mathcal{I}_C$ at time $last(Executed)$, or

(2) to assign to each proposition p a truth value if $(P?, last(Executed)) \in Executed$, where we represent such assignments as a set $\mathcal{I}_B$ of pairs (p, b) with $p \in \mathcal{P}$ and $b \in \{\top, \bot\}$, or

(3) to do both actions.

Thus, we denote an $\mathcal{IR}$ for `env` by a pair $(\mathcal{I}_C, \mathcal{I}_B)$. The *set of all instantaneous reactions* is denoted by $\mathcal{IR}^*$.

We now define how the outcome of the interplay between $\Delta_{\texttt{env}}$ and $\Delta_{\texttt{ctrl}}$ is handled. We start from the partial outcome that neglects $\mathcal{IR}$, and then we build the full outcome on top of the partial one considering $\mathcal{IR}$.

Definition 6 (Partial Outcome)**.** Let $\mathcal{PS} = (Executed, Assigned)$ be a locally consistent partial schedule; $\Delta_{\texttt{ctrl}}$ be an RTED for `ctrl` and $\Delta_{\texttt{env}}$ be an RTED for `env`. We model the *partial outcome* of $\Delta_{\texttt{ctrl}}$ and $\Delta_{\texttt{env}}$ as a mapping $PO\colon \mathcal{PS}^* \times \Delta^*_{\texttt{ctrl}} \times \Delta^*_{\texttt{env}} \to \mathcal{PS}^*$ neglecting any instantaneous reaction. We have four possible cases:

(1) $PO(\mathcal{PS}, \texttt{wait}, (t, Contingent)) = Executed \cup \{(C, t) \mid C \in Contingent\}$.

(2) $PO(\mathcal{PS}, (t_1, NonContingent), (t_2, Contingent)) = Executed \cup \{(C, t_2) \mid C \in Contingent\}$ if $t_2 < t_1$.

(3) $PO(\mathcal{PS}, (t, NonContingent), \texttt{wait}) = Executed \cup \{(X, t) \mid X \in NonContingent\}$.

(4) $PO(\mathcal{PS}, (t_1, NonContingent), (t_2, Contingent)) = Executed \cup \{(X, t_1) \mid X \in NonContingent\}$ if $t_1 \leq t_2$.

(1) says that `env` executes the time points in *Contingent* at time t if `ctrl` decides to do nothing. (2) says that `env` can execute the time points in *Contingent* because `env` decided to do so before `ctrl` executes his ($t_2 < t_1$). (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 as explained in the following definition.

Definition 7 (Full Outcome)**.** Let $PO(\mathcal{PS}, \Delta_{\texttt{ctrl}}, \Delta_{\texttt{env}})$ be a partial outcome and let $\mathcal{IR} = (\mathcal{I}_C, \mathcal{I}_B)$ be an instantaneous reaction. We model the *full outcome* as a mapping $FO\colon \mathcal{PS}^* \times \Delta^*_{\texttt{ctrl}} \times \Delta^*_{\texttt{env}} \times \mathcal{IR}^* \to \mathcal{PS}^*$ and we define it the same way we did for $PO(\mathcal{PS}, \Delta_{\texttt{ctrl}}, \Delta_{\texttt{env}})$ except that in cases (3) and (4) *Executed* is augmented with $\{(C, t) \mid C \in \mathcal{I}_C\}$ and *Assigned* is augmented with $\{(p, b) \mid (p, b) \in \mathcal{I}_B\}$. Either way $t = last(Executed)$.

The full outcome says how $\mathcal{PS}$ evolves according to the interplay of the RTEDs for `ctrl` and `env`, and `env`'s instantaneous reactions.

Given a partial schedule $\mathcal{PS}$, there are many possible RTEDs both for `ctrl` and for `env`. A *strategy* fixes `ctrl`'s RTEDs.

Definition 8 (RTED-based strategy)**.** An *RTED-based strategy for* `ctrl` is a mapping $\sigma_{\texttt{ctrl}} \colon \mathcal{PS}^* \to \Delta^*_{\texttt{ctrl}}$ from locally consistent partial schedules to RTEDs, whereas an *RTED-based strategy for* `env` is a mapping $\sigma_{\texttt{env}} \colon \mathcal{PS}^* \to \Delta^*_{\texttt{env}} \times \mathcal{IR}^*$ from locally consistent schedules to RTEDs and instantaneous reactions, respectively.

Definition 9 (One Step and Terminal Outcome)**.** The *one-step outcome* of the game played by `ctrl` and `env` is $FO^1(\mathcal{PS}, \sigma_{\texttt{ctrl}}, \sigma_{\texttt{env}}) = FO(\mathcal{PS}, \sigma_{\texttt{ctrl}}(\mathcal{PS}), v_1, v_2)$, where $\sigma_{\texttt{env}}(\mathcal{PS}) = (v_1, v_2)$. The *terminal outcome* $FO^*(\sigma_{\texttt{ctrl}}, \sigma_{\texttt{env}})$ is the complete schedule that results from the following recursive definition:

$$\begin{cases} \mathcal{PS}_0 = (\emptyset, \emptyset) \\ \mathcal{PS}_{i+1} = FO^1(\mathcal{PS}_i, \sigma_{\texttt{ctrl}}, \sigma_{\texttt{env}}) \end{cases}$$

Given the execution semantics for CSTNUs defined above, we can now provide the definition of dynamic controllability.

Definition 10 (Dynamic controllability of a CSTNU)**.** A CSTNU is *dynamically controllable (DC)* if there exists an RTED-based strategy $\sigma_{\texttt{ctrl}}$ such that for all RTED-based strategies $\sigma_{\texttt{env}}$, the variable assignments (X, k) in the complete schedule $FO^*(\sigma_{\texttt{ctrl}}, \sigma_{\texttt{env}})$ satisfy all constraints in $\mathcal{C}$ whose labels are implied by the (complete) scenario ℓ_{cps} which defines the truth value of all relevant propositions.

3.2 Dynamic Controllability of CSTNUs via Timed Game Automata

The *DC-checking* problem is the problem of deciding whether an arbitrary CSTNU is DC. We can answer the DC-checking problem by using *sound* and *complete* algorithms based on *Timed Game Automata (TGAs)* (Cimatti et al., 2014, 2016). We model this checking as a two-player game between `ctrl` and `env` according to the semantics we gave in Section 3.1.

The purpose of `ctrl` is to reach a specific location as soon as all time points have been executed and all constraints have been satisfied with respect to the specific scenario, whereas `env`'s goal is to prevent `ctrl` from entering that location. If `ctrl` wins, the network is DC, otherwise it is not.

In the following, we present some preliminary notions about TGAs and how to encode CSTNUs into TGAs. The DC-checking problem in a CSTNU is shown to be equivalent to the reachability problem in the corresponding TGA.

A *Finite Automaton* is defined as a tuple $\langle S, \rightarrow \rangle$, where S is a finite set of states and $\rightarrow$ is a finite set of labeled transitions. S always contains both a *starting state* and a subset of *final states*. Each transition specifies a legal move from one state to another (Hopcroft & Ullman, 1979).

A *Timed Automaton (TA)* refines a Finite Automaton by adding real-valued *clocks* and *clock constraints*. All clocks increase at the uniform rate keeping track of the time with respect to a fixed global time frame. Clocks are fictitious, i.e., invented to express the timing properties of the system. More formally,

Definition 11 (Timed Automaton, Alur & Dill, 1994)**.** A *Timed Automaton (TA)* is a tuple $\langle Loc, Act, \mathcal{X}, \rightarrow, Inv \rangle$, where

- *Loc* is a finite set of *locations* (with L_0 set as the initial one).

- *Act* is a finite set of *actions*. They are used as transition labels (they can be viewed as input symbols).
- $\mathcal{X}$ is a finite set of *real-valued clocks*.
- $\rightarrow \subseteq Loc \times \mathcal{H}(\mathcal{X}) \times Act \times 2^{\mathcal{X}} \times Loc$ is the transition relation. An edge (L_i, G, A, R, L_j) represents a transition from location L_i to location L_j realizing action A. $G \in \mathcal{H}(\mathcal{X})$ is a *guard* consisting on a conjunction of clock constraints having the form $x \sim k$ or $y - x \sim k$ with $x, y \in \mathcal{X}$, $k \in \mathbb{N}$ and $\sim \in \{<, \leq, =, >, \geq\}$.[3] If the values of the clocks satisfy the guard, the transition can be taken. The set $R \subseteq 2^{\mathcal{X}}$ specifies which clocks have to be reset (i.e., set to 0) whenever the transition is taken.
- Inv: $Loc \rightarrow \mathcal{H}(\mathcal{X})$ is a function assigning an *invariant* (i.e., a conjunction of clock constraints) to each location. An invariant is a condition under which the automaton may stay in that location.

Figure 3a shows an example of a TA. L_0 is the initial location. The TA has one clock `cX` set to 0 upon the start of an execution and it is allowed to remain in L_0 while $\texttt{cX} \leq 3$, then it has to leave the location. The `pass` transition can be taken whenever $\texttt{cX} \geq 1$ and along with $Inv(L_0)$ ensures that the TA will enter L_1 at any instant such that $1 \leq \texttt{cX} \leq 3$. When `pass` is taken, `cX` resets to 0. After that, the `gain` transition may be taken as soon as $\texttt{cX} \geq 5$. If taken, the TA gets back to L_0 and `cX` resets to 0. If not, the TA may remain in L_1 forever.

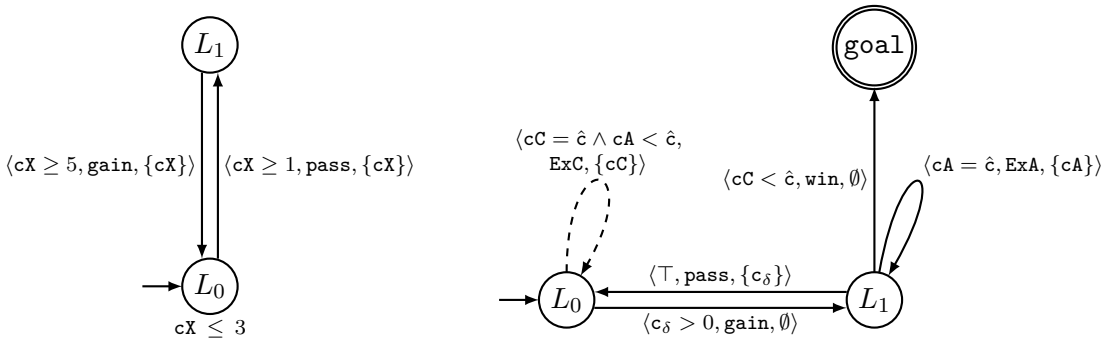
Definition 12 (Timed Game Automaton, Maler, Pnueli, & Sifakis, 1995)**.** A *Timed Game Automaton (TGA)* extends a TA by partitioning the set of transitions into *controllable* and *uncontrollable* ones. Uncontrollable transitions have priority over controllable ones.

In other words, if during an execution there are a set of controllable and a set of uncontrollable transitions that can be executed, then the uncontrollable ones are executed first and might prevent the controllable ones from being taken.

A TGA models a two-player timed game between a controller and the environment. The controller is assigned to controllable transitions, whereas the environment is assigned to uncontrollable ones. Moreover, in a TGA a location can be labeled as *urgent* to express that an enabled transition from the location must be taken immediately on entering.

Figure 3b shows an example of a TGA. The TGA has four clocks $\hat{\texttt{c}}, \texttt{c}_\delta, \texttt{cA}$ and `cC`. Solid arrows represent controllable transitions, whereas dashed arrows uncontrollable ones. The initial location is L_0, and `goal` is the location that `ctrl` must reach in order to win the game. Consider the following possible run. When all clocks are equal to 5, the `gain` transition is taken and the current location changes to L_1. At the same time, 5, the transition $\langle L_1, \texttt{cA} = \hat{\texttt{c}}, \texttt{ExA}, \{\texttt{cA}\}, L_1\rangle$ is taken resulting in the reset of `cA`. After that, the `pass` transition is taken always at time 5. Therefore, the current location becomes L_0 and the clock $\texttt{c}_\delta$ is reset to 0. At time 6, both the `ExC` and `gain` transitions are enabled and `ctrl` decides to take `gain`. At the same time, `env` decides to take `ExC`. Since `ExC` has priority, it executes first and, therefore, the clock `cC` is reset to 0. Then, the `gain`

3. The original definition allows one to consider the $\mathbb{Q}$ set as constant domain. However, Alur and Dill (1994) show that it is possible to consider $\mathbb{N}$ without loss of generality.



(a) Timed automaton (b) Timed game automaton

Figure 3: Examples of a) TA and b) TGA

transition can be taken at the same time 6 or later. After `gain`, the `win` transition can be taken to enter the `goal` location. For this TGA there does not exist a winning control strategy as the environment controlling the uncontrollable transitions can always refuse to take the uncontrollable transition. If it decides to do so, `cC` will never be reset, preventing the controller to take the `win` transition.

In what follows, we provide an improvement of the encoding given by Cimatti et al. (2016) from a CSTNU into a suitable TGA such that the CSTNU is DC if and only if `ctrl` can always win the game in the corresponding TGA by reaching the location `goal`. The peculiar property of Cimatti et al.'s encoding is that controllable transitions model the actions of environment, whereas uncontrollable transitions model the actions of the controller. This is due to the fact that the semantics given in terms of RTEDs for CSTNUs imposes that the controller's moves must go first. In what follows, $\hat{\mathtt{c}}$ is a clock representing the global time and it is never reset, whereas $\mathtt{c}_\delta$ is a special clock regulating the interplay between the game.

Figure 4 depicts the TGA corresponding to the CSTNU shown in Figure 2. The core of the TGA obtained by the CSTNU-TGA encoding always consists of three locations:

- L_0 (initial) is the location where `env` can take the controllable transitions modeling the occurrence of contingent time points and truth value assignments.
- L_1 (urgent) is the location where `ctrl` can take transitions for executing non-contingent time points.
- `goal` is the final location that, if reached (by `ctrl`), implies DC of the initial CSTNU.

We model the interplay between `ctrl` and `env` with a clock $\mathtt{c}_\delta$ and two transitions:

1. $(L_1, \top, \mathtt{pass}, \{\mathtt{c}_\delta\}, L_0)$
2. $(L_0, \mathtt{c}_\delta > 0, \mathtt{gain}, \emptyset, L_1)$

The first transition guarantees that, after some action(s) of `ctrl`, the environment can react immediately ($\top$ shortens the absence of any clock constraint and it is always interpreted

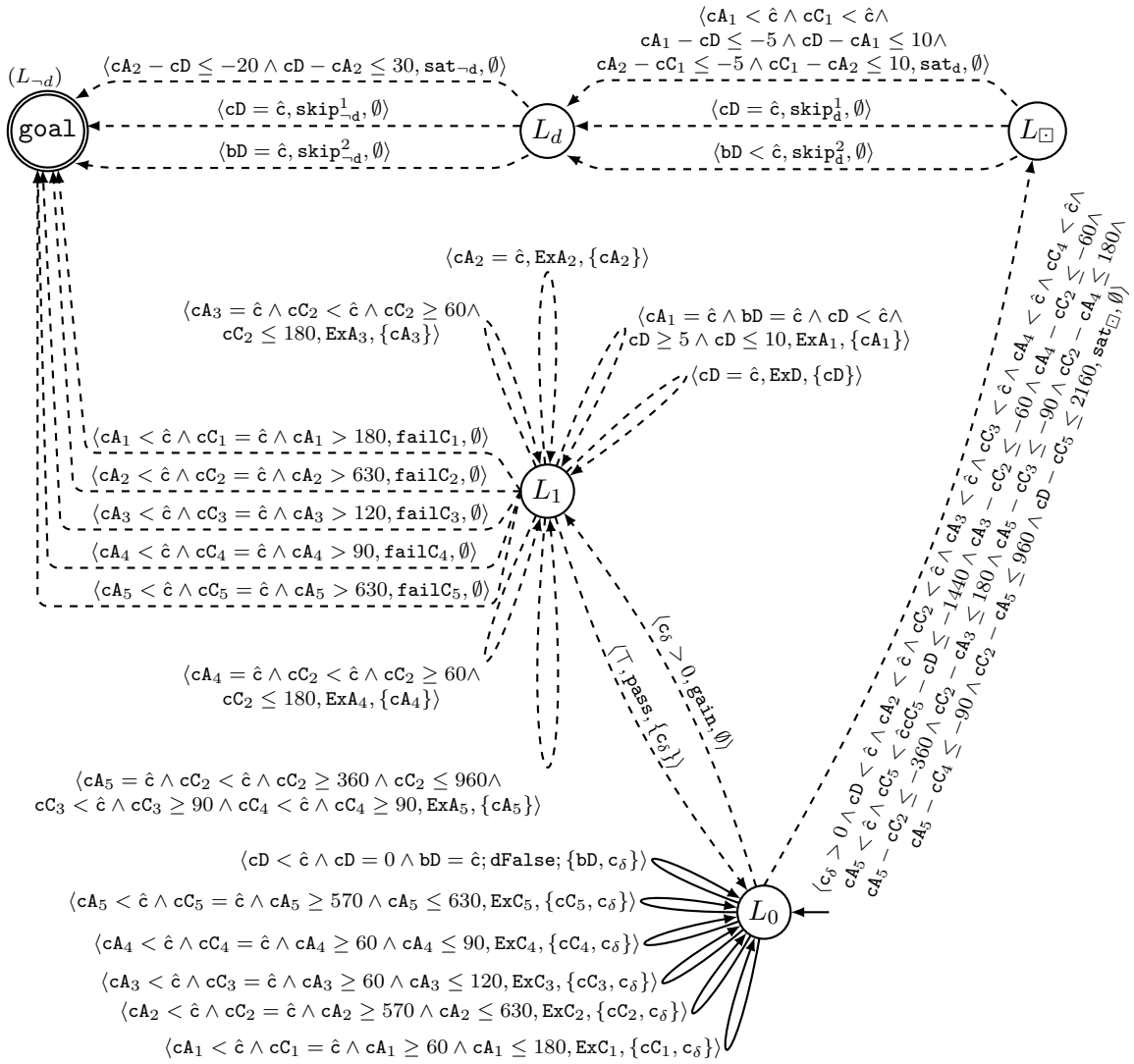


Figure 4: TGA encoding the CSTNU in Figure 2. L_0 is the initial location, L_1, $L_\boxdot$, L_d, goal are urgent. Solid (resp., dashed) edges model controllable (resp., uncontrollable) transitions.

true). The second one guarantees that $\texttt{ctrl}$ can react to an action of $\texttt{env}$ only after a positive delay $\epsilon > 0$ (modeled by $\texttt{c}_\delta > 0$).

We associate a clock $\texttt{bP}$ to each propositional letter p. If $\texttt{bP} = \hat{\texttt{c}}$ it means that $p = \top$, whereas if $\texttt{bP} < \hat{\texttt{c}}$ that $p = \bot$ (e.g., $\texttt{bD}$ in Figure 4).

We associate a clock $\texttt{cX}$ to each time point $X \in \mathcal{T}$. If $\texttt{cX} = \hat{\texttt{c}}$, it means that X has not been executed yet, whereas if $\texttt{cX} < \hat{\texttt{c}}$, it means that X was executed at time $\hat{\texttt{c}} - \texttt{cX}$. We reset the clocks associated to time points at most once. Therefore, we model the execution of a time point X by means of an *uncontrollable* self-loop transition at location L_1

$$(L_1, \texttt{cX} = \hat{\texttt{c}}, \texttt{ExX}, \{\texttt{cX}\}, L_1)$$

This transition can be executed only when $\texttt{cX}$ is equal to $\hat{\texttt{c}}$ (i.e., X has not been executed yet). The transition resets $\texttt{cX}$ fixing forever the time instant in which X was executed. A first improvement at this stage can be done considering also label honesty and conditions on predecessors in the transition guards (Zavatteri, 2017) in order to prevent the TGA from exploring impossible runs during the controller synthesis phase. For example, consider A_1 in Figure 2 and its label $L(A_1) = d$. Following the time point label honesty property for CSTNUs, we can extend the guard of $\texttt{ExA}_1$ by appending $\texttt{bD} = \hat{\texttt{c}} \wedge \texttt{cD} < \hat{\texttt{c}} \wedge \texttt{cD} > 0$ to model that A_1 must be executed if only if $d = \top$ (i.e., $\texttt{bD} = \hat{\texttt{c}}$), which also implies that A_1 must be executed after $D?$ (i.e., $D?$ have been executed ($\texttt{cD} < \hat{\texttt{c}}$)) and a positive amount of time ϵ has elapsed ($\texttt{cD} > 0$).

Definition 13 (Encoding time point label honesty)**.** A *label encoder* is a mapping $L_{enc} \colon \mathcal{T} \to \mathcal{H}(\mathcal{X})$ translating the label of a time point into the equivalent clock constraint $L_{enc}(X)$ encoding all literals containing propositions. Formally,

$$L_{enc}(X) \colon \bigwedge_{p \in L(X)} (\texttt{bP} = \hat{\texttt{c}} \wedge \texttt{cP} < \hat{\texttt{c}} \wedge \texttt{cP} > 0) \bigwedge_{\neg q \in L(X)} (\texttt{bQ} < \hat{\texttt{c}} \wedge \texttt{cQ} < \hat{\texttt{c}} \wedge \texttt{cQ} > 0)$$

We now focus on constraints. Consider the requirement link $D? \xrightarrow{[5,10],\, d} A_1$ in the CSTNU depicted in Figure 2. Such a constraint says that A_1 must be executed after 5 and within 10 since $D?$. This requirement link has also an important characteristic: $L(A_1)$ coincides with the label of the link. Therefore, whenever A_1 is executed, the constraint must hold. Thus, we extend the guard of $\texttt{ExA}_1$ by further appending $\texttt{cD} < \hat{\texttt{c}} \wedge \texttt{cD} \geq 5 \wedge \texttt{cD} \leq 10$, where the new conjuncts say that $D?$ has already been executed ($\texttt{cD} < \hat{\texttt{c}}$) and $A_1 - D? \in [5, 10]$ ($\texttt{cD} \geq 5 \wedge \texttt{cD} \leq 10$). Since clocks are reset upon the execution of time points we just need to measure the elapsed time since the reset of $\texttt{cD}$ to make sure that once we reset $\texttt{cA}_1$ the constraint holds (see also the modeling of contingent links that we discuss below). More formally:

Definition 14 (Encoding predecessors)**.** Given a CSTNU, a *predecessor* of a time point $Y \in \mathcal{T}$ is a time point $X \in \mathcal{T}$ such that there exists a constraint $(X - Y \leq -x, L(Y)) \in \mathcal{C}$, where $x > 0$. $\Pi : \mathcal{T} \to 2^{\mathcal{T}}$ returns the predecessors of a time point and it is formalized as

$$\Pi(Y) = \{X \mid (X - Y \leq -x, \ell) \in \mathcal{C} \wedge x > 0 \wedge \ell = L(Y)\}$$

A *predecessor encoder* is a mapping $\Pi_{enc} \colon \mathcal{T} \to \mathcal{H}(\mathcal{X})$ translating each $X \in \Pi(Y)$ (along with its temporal bounds) into an equivalent clock constraint as follows:

$$\Pi_{enc}(Y) = \bigwedge_{X \in \Pi(Y)} \texttt{cX} < \hat{\texttt{c}} \wedge \texttt{cX} \geq x \wedge \texttt{cX} \leq y\,,$$

where $\mathtt{cX} \geq x$ models $(X - Y \leq -x, L(Y))$ and $\mathtt{cX} \leq y$ models $(Y - X \leq y, L(Y))$ (if any).

Therefore, for each non-contingent time point X, the (improved) guard of $\mathtt{ExX}$ becomes

$$(L_1, \mathtt{cX} = \hat{\mathtt{c}} \wedge L_{enc}(X) \wedge \Pi_{enc}(X), \mathtt{ExX}, \{\mathtt{cX}\}, L_1)$$

Figure 4 contains 7 transitions ($\mathtt{ExD}, \mathtt{ExA}_1, \mathtt{ExA}_2, \mathtt{ExA}_3, \mathtt{ExA}_4$ and $\mathtt{ExA}_5$) modeling the execution (by $\mathtt{ctrl}$) of the non-contingent time points $D?$, A_1, A_2, A_3, A_4 and A_5 of the original CSTNU in which we enforced label honesty and predecessors according to (Zavatteri, 2017).

To allow the environment to flip the value of p (which is $\top$ by default), we add a controllable self loop at L_0

$$(L_0, \mathtt{cP} < \hat{\mathtt{c}} \wedge \mathtt{cP} = 0 \wedge \mathtt{bP} = \hat{\mathtt{c}}, \mathtt{pFalse}, \{\mathtt{bP}, \mathtt{c}_\delta\}, L_0)$$

saying that $\mathtt{env}$ might decide to assign $\bot$ to p resetting $\mathtt{bP}$. $\mathtt{env}$ may decide to do so only immediately after $\mathtt{ctrl}$ executed $P?$. That is, $\mathtt{cP} < \hat{\mathtt{c}}$ ($P?$ has been executed) and $\mathtt{cP} = 0$ (time has not elapsed). If $\mathtt{env}$ does not take such a transition, p will remain set to $\top$ forever. Furthermore, such a transition also resets $\mathtt{c}_\delta$ allowing $\mathtt{env}$ to take other controllable transitions. The complete management of a proposition p is realized by two transitions: one (uncontrollable) allowing $\mathtt{ctrl}$ to execute $P?$, and another one (controllable) allowing $\mathtt{env}$ to assign $\bot$ to p. In Figure 4, we label such a transition as $\mathtt{pFalse}$.

For each contingent link $(A, x, y, C) \in \mathcal{L}$, $\mathtt{env}$ can set the value of the contingent time point C by means of a controllable self-loop transition at L_0

$$(L_0, \mathtt{cA} < \hat{\mathtt{c}} \wedge \mathtt{cC} = \hat{\mathtt{c}} \wedge \mathtt{cA} \geq x \wedge \mathtt{cA} \leq y, \mathtt{ExC}, \{\mathtt{cC}, \mathtt{c}_\delta\}, L_0)$$

This transition can be taken only when time point A has been executed ($\mathtt{cA} < \hat{\mathtt{c}}$), time point C has not been executed ($\mathtt{cC} = \hat{\mathtt{c}}$), and the time elapsed since A was executed falls in $[x, y]$ (modeled as the clock constraint $\mathtt{cA} \geq x \wedge \mathtt{cA} \leq y$). The transition resets $\mathtt{cC}$ fixing the time in which C was executed and, again, $\mathtt{c}_\delta$ to allow other possible $\mathcal{IR}s$. Differently from truth-value assignments, $\mathtt{env}$ has to take this transition. If it does not, $\mathtt{ctrl}$ can win the game by taking the uncontrollable transition

$$(L_1, \mathtt{cA} < \hat{\mathtt{c}} \wedge \mathtt{cC} = \hat{\mathtt{c}} \wedge \mathtt{cA} > y, \mathtt{failC}, \emptyset, \mathtt{goal})$$

once it gets back to L_1. Indeed, this last transition says that the activation time point has been executed ($\mathtt{cA} < \hat{\mathtt{c}}$), the contingent time point has not be set ($\mathtt{cC} = \hat{\mathtt{c}}$) and it is currently out of its range of allowed values ($\mathtt{cA} > y$). For any contingent link, there are two transitions to manage the contingent time point: one to execute C and another one to force such an assignment. We call this latter transition a *fail* transition. In general, fail transitions for contingent links are always necessary because, otherwise, $\mathtt{env}$ would always be able to prevent $\mathtt{ctrl}$ from winning the game just "waiting forever" in L_0.

In Figure 4, there are 5 transitions for the execution of C_1, C_2, C_3, C_4 and C_5 ($\mathtt{ExC}_1$, $\mathtt{ExC}_2$, $\mathtt{ExC}_3$, $\mathtt{ExC}_4$, $\mathtt{ExC}_5$) and 5 to handle their failure ($\mathtt{failC}_1$, $\mathtt{failC}_2$, $\mathtt{failC}_3$, $\mathtt{failC}_4$, $\mathtt{failC}_5$).

We model the winning conditions of the game as a winning path of $n+1$ urgent locations $L_\boxdot \rightsquigarrow L_{\ell_1} \rightsquigarrow \cdots \rightsquigarrow L_{\ell_n} (= \mathtt{goal})$ going from L_0 to $\mathtt{goal}$, where n is the number of possible

distinct labels labeling time points and constraints. The winning path serves to verify that all relevant time points have been executed and all constraints involving them have been satisfied. The first location in this path is always $L_{\boxdot}$, reachable as soon as $\mathtt{c}_\delta > 0$ and all time points having empty label have been executed satisfying all constraints having empty label. Then, the remaining locations are sequentially connected through sets of transitions. Each set of transitions between two consecutive locations $L_{\ell_{i-1}}, L_{\ell_i}$ represents the conditional meta constraint "if the current scenario is associated to ℓ_i, then all time points labeled by ℓ_i must have been executed and all constraints labeled by ℓ_i must be satisfied". Such a conditional meta constraint can be represented as a disjunction of conjunctive expressions involving clocks and each single disjunct can be represented by a transition from $L_{\ell_{i-1}}$ to L_{ℓ_i} (Cimatti et al., 2016).
Therefore, each set of transitions between two consecutive locations represents two cases:

1. if $\ell_{cps} \not\Rightarrow \ell_i$, then at least one transition of the set allows `ctrl` to move to the next location, or

2. if $\ell_{cps} \Rightarrow \ell_i$, then exactly one transition allows `ctrl` to move to the next location if and only if all time points labeled by ℓ_i have been executed and all the constraints labeled by ℓ_i have been satisfied.

In Figure 4, the winning path consists of 3 locations only: $L_{\boxdot}$, L_d and $L_{\neg d}$ ($L_{\neg d}$ coincides with `goal`). There is a set of 3 transitions connecting $L_{\boxdot}$ to L_d and another set of 3 transitions between L_d and `goal`. We give an example for the set of transitions going from L_d to `goal`.

The conditional meta constraint for generating such a set of transitions is: "If D? has been executed and d was assigned $\top$, then all time points labeled by d must have been executed and all constraints labeled by d must have been satisfied". In symbols:[4]

$$(\mathtt{cD} < \hat{\mathtt{c}} \wedge \mathtt{bD} = \hat{\mathtt{c}}) \Rightarrow (\mathtt{cA}_1 < \hat{\mathtt{c}} \wedge \mathtt{cC}_1 < \hat{\mathtt{c}} \wedge \mathtt{cA}_1 - \mathtt{cD} \leq -5 \wedge \mathtt{cD} - \mathtt{cA}_1 \leq 10 \wedge \mathtt{cA}_2 - \mathtt{cC}_1 \leq -5 \wedge \mathtt{cC}_1 - \mathtt{cA}_2 \leq 10)$$

However, TGAs don't allow logical implications, disjunctions or negations of clock constraints in the guards. If a guard of a transition contains implications or disjunctions or negations of clock constraints, then it must be transformed in the disjunctive normal form and, then, the transition containing such a guard has to be represented as a set of parallel transitions, each containing as a guard a disjunct of the transformed guard. Therefore, we rewrite the previous meta conditional constraint as

$$\neg(\mathtt{cD} < \hat{\mathtt{c}} \wedge \mathtt{bD} = \hat{\mathtt{c}}) \vee$$
$$(\mathtt{cA}_1 < \hat{\mathtt{c}} \wedge \mathtt{cC}_1 < \hat{\mathtt{c}} \wedge \mathtt{cA}_1 - \mathtt{cD} \leq -5 \wedge \mathtt{cD} - \mathtt{cA}_1 \leq 10 \wedge \mathtt{cA}_2 - \mathtt{cC}_1 \leq -5 \wedge \mathtt{cC}_1 - \mathtt{cA}_2 \leq 10)$$

4. For each $l \leq Y - X \leq u \in \mathcal{C}$, the corresponding guard is $l \leq (\hat{\mathtt{c}} - \mathtt{cY}) - (\hat{\mathtt{c}} - \mathtt{cX}) \leq u$, which simplifies in $\mathtt{cX} - \mathtt{cY} \geq l \wedge \mathtt{cX} - \mathtt{cY} \leq u$ as $\forall X \in \mathcal{T}$. $(\hat{\mathtt{c}} - \mathtt{cX})$ is the time in which X was executed.

which simplifies to

$$(\mathtt{cD} = \hat{\mathtt{c}}) \vee$$
$$(\mathtt{bD} < \hat{\mathtt{c}}) \vee$$
$$(\mathtt{cA}_1 < \hat{\mathtt{c}} \wedge \mathtt{cC}_1 < \hat{\mathtt{c}} \wedge \mathtt{cA}_1 - \mathtt{cD} \leq -5 \wedge \mathtt{cD} - \mathtt{cA}_1 \leq 10 \wedge \mathtt{cA}_2 - \mathtt{cC}_1 \leq -5 \wedge \mathtt{cC}_1 - \mathtt{cA}_2 \leq 10)$$

Note that we could have simplified the first two disjuncts to $\mathtt{cD} \geq \hat{\mathtt{c}}$ and $\mathtt{bD} \geq \hat{\mathtt{c}}$. However, we prefer to adhere to the standard semantics in which $\mathtt{cD} = \hat{\mathtt{c}}$ means that $D?$ has not been executed and $\mathtt{bD} = \hat{\mathtt{c}}$ means that d is $\top$.

Figure 4 represents these three disjuncts as a set of 3 transitions: $\mathtt{sat_d}$ whose guard is $(\mathtt{cA}_1 < \hat{\mathtt{c}} \wedge \mathtt{cC}_1 < \hat{\mathtt{c}} \wedge \mathtt{cA}_1 - \mathtt{cD} \leq -5 \wedge \mathtt{cD} - \mathtt{cA}_1 \leq 10 \wedge \mathtt{cA}_2 - \mathtt{cC}_1 \leq -5 \wedge \mathtt{cC}_1 - \mathtt{cA}_2 \leq 10)$, $\mathtt{skip}_\mathtt{d}^1$ whose guard is $\mathtt{cD} = \hat{\mathtt{c}}$ and $\mathtt{skip}_\mathtt{d}^2$ whose guard is $\mathtt{bD} < \hat{\mathtt{c}}$. We generate the set of transitions going from $L_{\boxdot}$ to L_d similarly.

Without loss of generality, this encoding does not allow the execution of time points at time 0. Indeed, any run starts at L_1 and can enter L_0 only after a positive amount of time, $c_\delta > 0$. Such a limitation is not really meaningful, as it is always possible to translate the obtained schedule into a corresponding one where all time assignments are shifted of $-c_\delta$.

In (Cimatti et al., 2014, 2016) the authors showed that the DC-checking of a CSTNU is equivalent to (model) checking the corresponding TGA and looking for a control strategy for `env` to always prevent `ctrl` from reaching `goal`. If such a strategy exists, then the original CSTNU is not DC, otherwise it is.

4. CSTNUs with Resources (CSTNURs)

In this section, we propose *Conditional Simple temporal Networks with Uncertainty and Resources (CSTNURs)* as an extension of CSTNUs obtained by injecting:

- resources with associated temporal expressions,
- runtime resource constraints.

Informally, in a CSTNUR, a resource must be committed for executing a time point. Such a resource can be chosen from a set associated to the time point. Each resource is associated to a temporal expression representing its temporal availability during execution. A resource can be committed to execute a time point at a certain time instant t if and only if t satisfies the temporal expression associated to the resource. Moreover, the availability of a resource may be constrained by means of a special kind of constraints, *runtime resource constraints*. A runtime resource constraint further restricts the availability of a resource at runtime according to the execution time of previous time points or previous resource commitments.

4.1 Syntax of CSTNURs

In what follows, we introduce some preliminary notions before giving the formal definition of CSTNUR.

A *temporal expression* represents an assertion with respect to an implicit temporal instant and a possible time point; a temporal expression is useful to characterize the temporal availability of resources in a CSTNUR in a compact way.

Definition 15 (Temporal Expression)**.** A *temporal expression (TE)* τ is a (temporal) assertion defined according to the grammar:

$$\tau ::= \odot \mid \theta k \mid \theta X + k \mid \tau_1 \wedge \tau_2$$

where $\odot$ is the empty constraint, $\theta \in \{>, <, \geq, \leq, =\}$, $k \in \mathbb{N}$, X is a time point, and τ_1, τ_2 are two TEs. There are 4 types of TEs:

***Type 0*:** $\tau = \odot$ (empty).

***Type 1*:** $\tau = \theta k$ (constant)

***Type 2*:** $\tau = \theta X + k$ (relative to a time point X)

***Type 3*:** $\tau = \tau_1 \wedge \tau_2$ (conjunction)

The set of all possible TEs is denoted by $\mathcal{TE}$.

Every TE of Type 2, $\tau = \theta X + k$, is equivalent to a Type 1 once X has been executed. We determine the truth value of a TE τ, with respect to a particular instant t, by means of the following interpretation.

Definition 16 (TE interpretation)**.** The *interpretation of a TE* τ with respect to a temporal instant $t \in \mathbb{R}^{\geq 0}$ is defined as follows:

1. $t \models \odot$
2. $t \models \theta k$ iff $t \,\theta\, k$, where $k \in \mathbb{N}$ and $\theta \in \{>, <, \geq, \leq, =\}$.
3. $t \models \theta X + k$ iff $t \,\theta\, (t_X + k)$, where $k \in \mathbb{N}$, $\theta \in \{>, <, \geq, \leq, =\}$, and t_X is the time at which X was executed. If X is unexecuted when expression has to be evaluated, then t_X is assumed to be $+\infty$.
4. $t \models \tau_1 \wedge \tau_2$ iff $t \models \tau_1$ and $t \models \tau_2$.

By using temporal expressions, it is possible to represent the notion of *availability* of resources at runtime in a compact way.

Definition 17 (Temporal availability)**.** Given a set of resources $\mathcal{R}$, a set of time points $\mathcal{T}$ and the set of all possible temporal expressions $\mathcal{TE}$, the *temporal availability* of resources is a pair (RA, RE), where:

- $RA \subseteq \mathcal{R} \times \mathcal{T}$ determines which resources can be committed for each time point. We write $\mathcal{R}(X) = \{r \mid (r, X) \in RA\}$ to represent the *resources* committable for time point X and we impose that for each contingent link $(A, x, y, C) \in \mathcal{L}$, $\mathcal{R}(A) = \mathcal{R}(C)$ and for each $X \in \mathcal{T}$, $\mathcal{R}(X) \neq \emptyset$.
- $RE\colon RA \to \mathcal{TE}$ represents the temporal expression associated to an element (r, X), where $r \in \mathcal{R}(X)$.

A resource $r \in \mathcal{R}(X)$ is *committable at time* t if $t \models RE(r, X)$, not committable, otherwise.

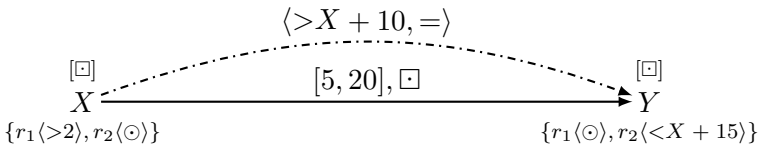


Figure 5: An augmented CSTNU. Each time point has an associated set consisting of elements having the form $r\langle\tau\rangle$ meaning $(r, X) \in RA \wedge RE(r, X) = \tau$. The dash-dotted edge represents a Runtime Resource Constraint (RRC), introduced in Definition 18.

Figure 5 shows an augmented CSTNU consisting of two time points X and Y connected by two edges. Each time point has two labels: one representing the CSTNU propositional label associated to the time point (in the figure both time points have $\boxdot$ as propositional label), and the other representing the set of committable resources for the time point. To simplify the notation, the expression $r\langle\tau\rangle$ associated to a time point X means $(r, X) \in RA$ and $RE(r, X) = \tau$. In details, for X, the expression $r_1\langle > 2\rangle$ means that r_1 can be committed to execute X only when the execution time is greater than 2. For the resource r_2, the temporal expression with respect to Y is $< X + 15$ meaning that r_2 can be committed to execute Y only if the execution time is smaller than $t_X + 15$, where t_X is the execution time of X.

Runtime resource constraints (RRCs) refine temporal expressions of resources by specifying further resource constraints involving multiple time points. Each RRC is defined between two time points, a *firing time point* and a *target time point*. When the firing time point is executed, the effect of the RRC is to append new TEs to the existing ones associated to the resources of the target time point (if such a time point is still unexecuted) considering the relation defined in the RRC. In this way, it is possible to adjust, for example, the committable resources of the target time point considering which resource was committed for the firing time point.

Definition 18 (Runtime Resource Constraint)**.** A *Runtime Resource Constraint (RRC)* is a 4-tuple $\langle X, \tau, Y, \rho\rangle$, where:

- $X, Y \in \mathcal{T}$ are the firing and target time points, respectively, such that $X \neq Y$, Y is non contingent, and $L(X)$ is consistent with $L(Y)$ (and with $L(W)$ if τ is a Type 2 TE involving time point W).
- $\tau \in \mathcal{TE}$ is a temporal expression.
- $\rho \subseteq \mathcal{R} \times \mathcal{R}$ is a binary relation over resources. As usual, $=$ shortens $\{(r, r) \mid r \in \mathcal{R}\}$, $\neq$ shortens $\{(r_1, r_2) \mid r_1, r_2 \in \mathcal{R} \wedge r_1 \neq r_2\}$, and $*$ shortens $\{(r_1, r_2) \mid r_1, r_2 \in \mathcal{R}\}$ (the universal relation).

We interpret each RRC $\langle X, \tau, Y, \rho\rangle$ as follows: when a resource r_X is committed to execute X, then τ is *instantaneously* appended to all temporal expressions of those resources r_Y committable for Y such that $(r_X, r_Y) \in \rho$. In symbols,

$$\forall r_Y \in \mathcal{R}(Y).\ (r_X, r_Y) \in \rho \implies RE(r_Y, Y) := RE(r_Y, Y) \wedge \tau\,,$$

where $r_X \in \mathcal{R}(X)$ is the resource committed for X. An RRC does not imply an execution order among time points. An RRC $\langle X, \tau, Y, \rho \rangle$ has effect on Y instantaneously, after the execution of X, providing that Y is still unexecuted. Therefore, if there is an RRC between X and Y and the two time points have to be executed at the same time, it is necessary to fix an execution order between them to decide whether the RRC applies. Moreover, if there is an RRC between a contingent time point C and a non-contingent time point X and the two time points occur at the same time t (because `env` decided to execute C after `ctrl` had decided to execute X at time t), then the RRC is ignored because C is assumed to be executed after X even if the two time points are executed at the same instant.

In Figure 5, the RRC $\langle X, >X+10, Y, = \rangle$, drawn as $X \xrightarrow{\langle >X+10, = \rangle} Y$, represents the fact that the resource committed for X can be committed for Y only if the execution time of Y is greater than 10 time units since X was executed. Note that the temporal constraint between X and Y allows the execution of Y just 5 time units after X. Suppose that r_2 is committed for X at time 1. $\mathcal{R}(Y)$ is instantaneously updated considering $\langle X, >X+10, Y, = \rangle$. Since the RRC becomes $\langle X, >11, Y, = \rangle$ at time 1 because $X = 1$, its TE part >11 is appended to all TEs associated to the resources satisfying ρ. In this case, it applies only to r_2 because ρ is '='. Therefore, the application of the RRC results in evolving the "state" of the temporal expressions of the resources in $\mathcal{R}(Y)$ as follows:

$$\overbrace{\{r_1\langle\odot\rangle, r_2\langle <X+15\rangle\}}^{\mathcal{R}(Y) \text{ when } t<1} \quad \rightsquigarrow \quad \overbrace{\{r_1\langle\odot\rangle, r_2\langle >11 \ \wedge \ <16\rangle\}}^{\mathcal{R}(Y) \text{ when } t\geq 1}$$

The delay allowed for executing Y after X is $[5, 20]$. If `ctrl` decides to fix the execution of Y at t' in $[5, 11]$ (respectively, $[16, 20]$), then the only committable resource is r_1 (respectively, r_1 and r_2).

Now, it is possible to give the formal definition of a CSTNUR putting together everything we have discussed so far.

Definition 19 (Conditional Simple Temporal Network with Uncertainty and Resources)**.** A *Conditional Simple Temporal Network with Uncertainty and Resources (CSTNUR)* is a tuple $\langle \mathcal{T}, \mathcal{P}, L, \mathcal{OT}, O, \mathcal{C}, \mathcal{L}, \mathcal{R}, RA, RE, \mathcal{RRC} \rangle$, where:

1. $\langle \mathcal{T}, \mathcal{P}, L, \mathcal{OT}, O, \mathcal{C}, \mathcal{L} \rangle$ is a CSTNU.
2. $\mathcal{R} = \{r_0, r_1, \dots\}$ is a finite set of resources.
3. The pair (RA, RE) specifies temporal availability according to Definition 17.
4. $\mathcal{RRC}$ is a set of runtime resource constraints according to Definition 18.

Figure 6 shows the CSTNUR extending the CSTNU in Figure 2 and considering resource specification. There are seven users, `Alice`, `Bob`, `Lila`, `Mike`, `Evie`, `Tim` (shortened as `a`, `b`, `l`, `m`, `e`, and `t`, respectively) and `wf`, where `wf` represents an internal agent, such as, for example, a workflow engine. $C_2 \xrightarrow{\langle \geq C_2+840, = \rangle} A_5$ (RRC_1) models a *temporal separation of duties (TSoD)* meaning that the *same* pilot (=) who executes C_2 (i.e., piloted the aircraft in the `OutFlight`) will return available to pilot again after 14 hours (FAA regulations). $A_3 \xrightarrow{\langle >C_3, = \rangle} A_4$ (RRC_2) and $A_4 \xrightarrow{\langle >C_4, = \rangle} A_3$ (RRC_3) model a "no multi-tasking" policy for

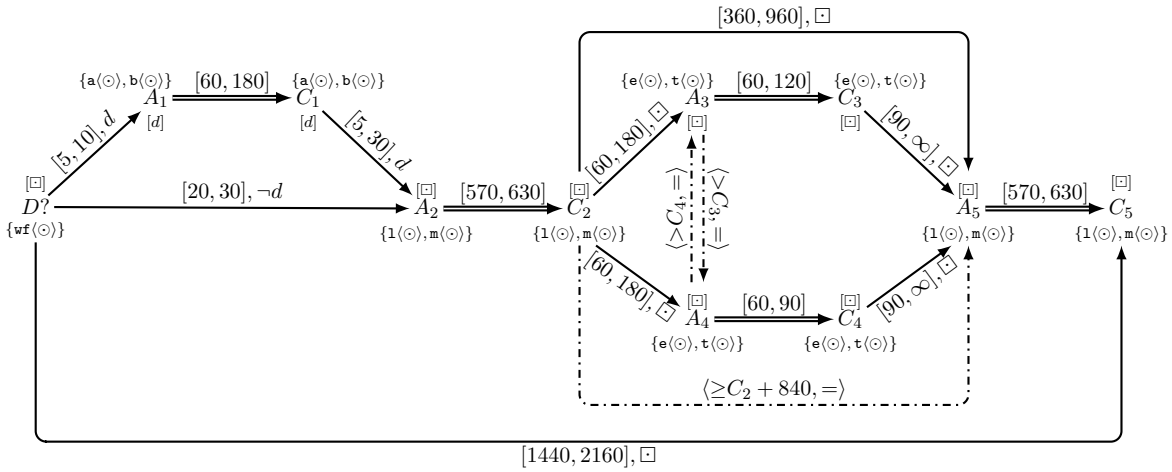


Figure 6: CSTNUR modeling the temporal plan in Figure 1 with access control. Dashed edges represent RRCs.

`SysCheck` and `SecCheck` requiring that either different resources are committed for those tasks when executed in parallel, or the same resource can be committed for both, provided that these tasks are executed sequentially. Indeed, RRC_2 specifies that the user who executes A_3 will return available for executing A_4 as soon as C_3 has executed. Likewise, RRC_3 specifies that the user who starts A_4 will become again available for executing A_3 as soon as C_4 has executed.

4.2 Execution Semantics of CSTNURs

In a CSTNUR, resources are committed to execute time points. Resources committed for contingent time points are the same that were committed for the corresponding activation time points. However, `env` is still free to schedule these time points when he wants.

Since CSTNURs extend CSTNUs, we still have that the truth values of propositions and the duration of contingent links are incrementally revealed over time as the corresponding observation and contingent time points are executed, respectively. Again, a *dynamic execution strategy* reacts to observations and contingent time points in real time also saying which resources are committed for which time points. A *viable* and *dynamic execution strategy* for a CSTNUR is a strategy executing all non-contingent time points such that all relevant constraints about temporal distances and resource commitments will be satisfied no matter which truth values for propositions and durations for contingent links are incrementally revealed over time. A CSTNUR with such a strategy is called *dynamically controllable*.

A more formal description of the execution semantics of CSTNURs can be given in terms of extended RTEDs. In what follows, we extend the RTEDs given for CSTNUs to also consider resources and RRCs. For a CSTNUR, `ctrl` seeks a strategy for scheduling all relevant non-contingent time points such that all relevant temporal constraints involving resources and time points are eventually satisfied no matter what `env` does.

A *partial schedule* for a CSTNUR is still a pair $\mathcal{PS} = (Executed, Assigned)$ but here *Executed* is a set of triples (r, X, t), where r is the resource committed for X at time t. Instead, *Assigned* remains the same. For each $X \in Executed_{\mathcal{T}}$, $time(X)$ still queries *Executed* to get information about when X was executed, whereas $res(X)$ does the same but with respect to the committed resource. $\mathcal{PS}$ is *locally consistent* if *Executed* satisfies all temporal constraints of the underlying CSTNU and for each $(r, X, t) \in Executed_{\mathcal{T}}$, $r \in \mathcal{R}(X)$ and $t \models RE(r, X)$, where $RE(r, X)$ is the temporal expression associated to the pair (r, X).[5] The set of all possible partial schedules remains represented by $\mathcal{PS}^*$.

In the following, we fully formalize the execution semantics of CSTNURs in terms of extended RTEDs for `env` and `ctrl`. Moreover, *NonContingent* is now a set of pairs (r, X), where r is the resource that `ctrl` wants to commit for X.

Definition 20 (RTED for `ctrl`)**.** An *RTED for the controller* `ctrl`, $\Delta_{\texttt{ctrl}}$, specifies which action has to be performed by `ctrl` during an execution (represented by $\mathcal{PS}$). It has two forms: `wait` or $(t, NonContingent)$.

- $\Delta_{\texttt{ctrl}} = \texttt{wait}$ is the same as that given in Definition 3.
- $\Delta_{\texttt{ctrl}} = (t, NonContingent)$ represents the conditional constraint: "if `env` does nothing *before* time t, then for each pair $(r, X) \in NonContingent$, commit the resource r to execute time point X at time t." Such a decision is applicable if and only if $t > last(Executed)$, *NonContingent* is a (non empty) *ordered* set of pairs (r_i, X_i) $i = 1, \ldots, k$, where r_i is a resource associated to X_i and committable at time t and X_i is a non-contingent unexecuted time point such that $\ell_{cps} \Rightarrow L(X_i)$.

Definition 21 (RTED for `env`)**.** An *RTED for the environment* `env`, $\Delta_{\texttt{env}}$, specifies which action has to be performed by `env` during an execution. It has two forms: `wait` or $(t, Contingent)$.

- $\Delta_{\texttt{env}} = \texttt{wait}$ is the same as that given in Definition 4.
- $\Delta_{\texttt{env}} = (t, Contingent)$ is the same as that given in Definition 4 committing $res(A)$ to execute C. In other words, such a decision is applicable if and only if $t > last(Executed)$, *Contingent* is a non-empty subset of pairs (r, C), where C is the contingent time point such that $(A, x, y, C) \in \mathcal{L}$ and $A \in Executed_{\mathcal{T}}$, $r = res(A)$ is the resource that was committed for A and $t \in [time(A) + x, time(A) + y]\}$.

$\Delta^*_{\texttt{ctrl}}$ and $\Delta^*_{\texttt{env}}$ still denote the sets of all RTEDs for `ctrl` and `env`. Furthermore, since the association of resources to contingent time points is implicit and the assignment of truth values to propositions does not involve resources, the instantaneous reactions definition is the same as the one given for CSTNUs in Definition 5.

We are now ready to extend the notion of the partial and full outcome between $\Delta_{\texttt{ctrl}}$ and $\Delta_{\texttt{env}}$.

5. Once r has been committed for X at time t, no RRC will ever be able to restrict r's associated temporal expression for X as RRCs only apply to unexecuted target time points. Therefore, the valuation of $t \models RE(u, X)$ will remain fixed forever.

Definition 22 (Partial Outcome)**.** Let $\mathcal{PS}$ be a locally consistent partial schedule. Let $\Delta_{\texttt{ctrl}}$ be an RTED for `ctrl` and $\Delta_{\texttt{env}}$ and RTED for `env`. We model the *partial outcome* of $\Delta_{\texttt{ctrl}}$ and $\Delta_{\texttt{env}}$ as a mapping $PO(\mathit{Executed}, \Delta_{\texttt{ctrl}}, \Delta_{\texttt{env}})$ neglecting any instantaneous reaction $\mathcal{IR}$. There are four possible cases:

(1) $PO(\mathit{Executed}, \texttt{wait}, (t, \mathit{Contingent})) = \mathit{Executed} \cup \{(\mathit{res}(A), C, t) \mid C \in \mathit{Contingent}\}$ Also, for any $\langle C, \tau, Y, \rho\rangle \in \mathcal{RRC}$ such that $Y \notin \mathit{Executed}_{\mathcal{T}}$ we have that $\forall r_Y \in \mathcal{R}(Y)$, $(\mathit{res}(A), r_Y) \in \rho \implies RE(r_Y, Y) := RE(r_Y, Y) \wedge \tau$.

(2) $PO(\mathit{Executed}, (t_1, \mathit{NonContingent}), (t_2, \mathit{Contingent})) = \mathit{Executed} \cup \{(\mathit{res}(A), C, t_2) \mid C \in \mathit{Contingent}\}$ if $t_2 < t_1$. Also, for any $\langle C, \tau, Y, \rho\rangle \in \mathcal{RRC}$ such that $Y \notin \mathit{Executed}_{\mathcal{T}}$ we have that $\forall r_Y \in \mathcal{R}(Y), (\mathit{res}(A), r_Y) \in \rho \implies RE(r_Y, Y) := RE(r_Y, Y) \wedge \tau$.

(3) $PO(\mathit{Executed}, (t, \mathit{NonContingent}), \texttt{wait}) = \mathit{Executed} \cup \{(r_i, X_i, t) \mid (r_i, X_i) \in \mathit{NonContingent}, \forall i = 1, \ldots, k\}$. Also, every time we add (r_i, X_i, t) to *Executed* we fire the related RRCs (if any). That is, for any $\langle X_i, \tau, Y, \rho\rangle \in \mathcal{RRC}$ such that $Y \notin \mathit{Executed}_{\mathcal{T}}$ we have that $\forall r_Y \in \mathcal{R}(Y), (r_i, r_Y) \in \rho \implies RE(r_Y, Y) := RE(r_Y, Y) \wedge \tau$.

(4) $PO(\mathit{Executed}, (t_1, \mathit{NonContingent}), (t_2, \mathit{Contingent})) =$
$\mathit{Executed} \cup \{(r_i, X_i, t) \mid (r_i, X_i) \in \mathit{NonContingent} \text{ for } i = 1, \ldots, k\}$ if $t_1 \leq t_2$. Again, every time we add (r_i, X_i, t) to *Executed* we fire the related RRCs (if any). That is, for any $\langle X_i, \tau, Y, \rho\rangle \in \mathcal{RRC}$ such that $Y \notin \mathit{Executed}_{\mathcal{T}}$ we have that $\forall r_Y \in \mathcal{R}(Y), (r_i, r_Y) \in \rho \implies RE(r_Y, Y) := RE(r_Y, Y) \wedge \tau$.

The explanations are similar to those given for Definition 3 considering resources who are now committed for time points. The definitions of full outcome and RTED-based strategies for `ctrl` and `env` are the same as those given for CSTNUs. The definition of dynamic controllability thus refines to:

Definition 23 (Dynamic Controllability of a CSTNUR)**.** A CSTNUR is *dynamically controllable (DC)* if there exists an RTED-based strategy $\sigma_{\texttt{ctrl}}$ such that for all RTED-based strategies $\sigma_{\texttt{env}}$, the variable assignments (r, X, k) in the complete schedule $FO^*(\sigma_{\texttt{ctrl}}, \sigma_{\texttt{env}})$ satisfy all temporal constraints of the underlying CSTNU, and each assignment (r, X, k) satisfies both $r \in \mathcal{R}(X)$ and $k \models RE(r, X)$.

5. Encoding CSTNURs into TGAs

In this section, we extend the encoding into TGAs given in Section 3.2 for CSTNUs in order to represent the DC checking of CSTNURs as a two-player game between `ctrl` and `env` according to the semantics we gave in Section 4.2.

We encode committable resources into dedicated clocks and resource commitments for time point executions considering RRCs into circular paths.

Such an encoding runs in polynomial time. We prove the correctness and complexity of such an encoding in Section 6.

5.1 Encoding Committable Resources into Dedicated Clocks

As we have already pointed out, in a CSTNUR resources are committed for time points according to the *RA* relation. Therefore, to model which resource has been committed for

which time point, we start by associating a dedicated clock $\mathtt{rX}$ to each element $(r, X) \in RA$ and interpreting the value of such clocks as follows ($\mathtt{cX}$ is the clock associated to X):

1. If $\mathtt{rX} = \mathtt{cX} = \hat{\mathtt{c}}$, it means that X has not been executed yet and r is committable for X.

2. If $\mathtt{rX} = \mathtt{cX} < \hat{\mathtt{c}}$, it means that X has been executed and r is not the resource committed for X.

3. If $\mathtt{rX} = \hat{\mathtt{c}} > \mathtt{cX}$, it means r was committed for X at time $\hat{\mathtt{c}} - \mathtt{cX}$.

4. If $\mathtt{rX} < \mathtt{cX} = \hat{\mathtt{c}}$, it means that X has not been executed yet and r cannot be committed for X because r is not available. In details, $\mathtt{rX}$ becomes less than $\mathtt{cX}$ by a reset action. The reset of $\mathtt{rX}$ occurs when it is necessary to prevent r from being committed for X since $\hat{\mathtt{c}} \not\models RE(r, X)$.

Differently from what we have done for clocks associated to time points, here $\mathtt{rX}$ clocks may be reset more than once. If r is committable for X and $\mathtt{rX}$ has never been reset, then r can be committed to execute X. To determine which resource was committed for a time point X, it suffices to check that $\mathtt{cX} < \hat{\mathtt{c}}$ and find the unique clock $\mathtt{rX}$ such that $\mathtt{rX} > \mathtt{cX}$. It is guaranteed that all other clocks $\mathtt{r}_j\mathtt{X}$ where $r_j \in \mathcal{R}(X)$ and $r_j \neq r$ are equal to $\mathtt{cX}$.

Getting back to our example, the relation RA specified for the CSTNUR in Figure 6 implies the following clocks.

- $\mathtt{wD}$ models $\mathcal{R}(D?) = \{\mathtt{wf}\}$
- $\mathtt{aA}_1$, $\mathtt{bA}_1$ model $\mathcal{R}(A_1) = \{\mathtt{Alice}, \mathtt{Bob}\}$
- $\mathtt{aC}_1$, $\mathtt{bC}_1$ model $\mathcal{R}(C_1) = \{\mathtt{Alice}, \mathtt{Bob}\}$
- $\mathtt{lA}_2$, $\mathtt{mA}_2$ model $\mathcal{R}(A_2) = \{\mathtt{Lila}, \mathtt{Mike}\}$
- $\mathtt{lC}_2$, $\mathtt{mC}_2$ model $\mathcal{R}(C_2) = \{\mathtt{Lila}, \mathtt{Mike}\}$
- $\mathtt{eA}_3$, $\mathtt{tA}_3$ model $\mathcal{R}(A_3) = \{\mathtt{Evie}, \mathtt{Tim}\}$
- $\mathtt{eC}_3$, $\mathtt{tC}_3$ model $\mathcal{R}(C_3) = \{\mathtt{Evie}, \mathtt{Tim}\}$
- $\mathtt{eA}_4$, $\mathtt{tA}_4$ model $\mathcal{R}(A_4) = \{\mathtt{Evie}, \mathtt{Tim}\}$
- $\mathtt{eC}_4$, $\mathtt{tC}_4$ model $\mathcal{R}(C_4) = \{\mathtt{Evie}, \mathtt{Tim}\}$
- $\mathtt{lA}_5$, $\mathtt{mA}_5$ model $\mathcal{R}(A_5) = \{\mathtt{Lila}, \mathtt{Mike}\}$
- $\mathtt{lC}_5$, $\mathtt{mC}_5$ model $\mathcal{R}(C_5) = \{\mathtt{Lila}, \mathtt{Mike}\}$

The following encoding aims to guarantee some properties that are necessary to state the equivalence between the CSTNUR DC checking problem and the reachability problem in the corresponding TGA. Before introducing such properties, we propose a compact notation about the clocks and state of $\mathcal{G}$. Let $\vec{c} = (\hat{\mathtt{c}}, \mathtt{c}_\delta, \mathtt{cX}, \ldots, \mathtt{cY}, \mathtt{bP}, \ldots, \mathtt{r}_0\mathtt{X}, \ldots, \mathtt{r}_0\mathtt{Y}, \ldots)$ be the vector containing all clocks of $\mathcal{G}$. A state of the TGA $\mathcal{G}$ is a pair $(L, \vec{c})$, where L is a location and $\vec{c}$ represents the values of all clocks. The properties that must be guaranteed in each state TGA $(L, \vec{c})$ corresponding to a locally consistent partial schedule of the considered CSTNUR are:

- $L = L_0$, $\mathtt{c}_\delta = 0$, $last(Executed) = \hat{\mathtt{c}}$.
- For each executed time point X, $time(X) = \hat{\mathtt{c}} - \mathtt{cX}$ and $res(X) = \mathtt{rX}$, where $\mathtt{rX}$ is the only $\mathtt{r}_i\mathtt{X}$ clock not reset.

- For each unexecuted time point X, $time(X) = \hat{c}$ and $res(X)$ is not defined (i.e., all $\mathtt{r}_i\mathtt{X} = \hat{c}$).
- For each executed observation time point $P?$, if $p = \top$ then $\mathtt{bP} = \hat{c}$, and if $p = \bot$ then $\mathtt{bP} < \hat{c}$. If the time point is not executed, the value of $\mathtt{bP}$ means nothing.

5.2 Encoding Resource Commitments into Circular Paths

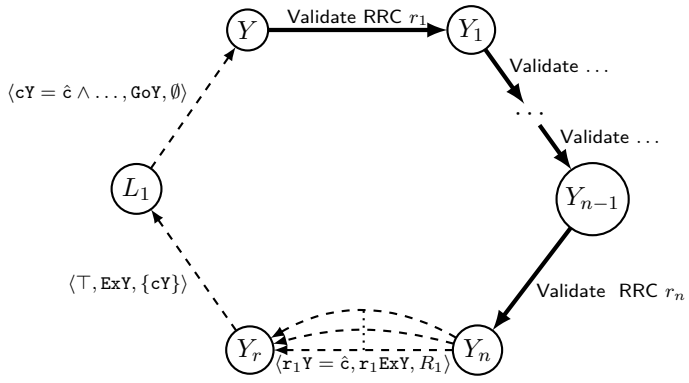
In the encoding for CSTNUs given in Section 3.1 the executions of time points are modeled as self-loop transitions at L_1. In CSTNURs, we must extend this part to model that resources are committed to execute time points. A resource r is committable to execute a time point Y at time t if $r \in \mathcal{R}(Y)$ and t satisfies its associated TE. The latter condition entails validating all fired RRCs (if any) targeting the time point we are trying to execute. We achieve all this as follows.

Instead of having a (possibly) exponential number of self-loop transitions modeling all possible executions with respect to all possible combinations of RRCs, we model the commitment of a resource r for a time point X by means of a *circular path* of *urgent locations* starting and ending at L_1 (see Figure 7a). All transitions involving these locations are uncontrollable. The first location has the same name of the time point to execute (i.e., Y). A run of the TGA enters Y if and only if the corresponding time point has not been executed yet (the guard is exactly the same of that given for the self-loop transitions for CSTNUs). Then, the run goes through a set of n locations $Y_1, \ldots, Y_n$, where n is the number of RRCs targeting Y. Moving from Y_{i-1} to Y_i means validating the i^{th} RRC targeting Y. In Figure 7a, each thick edge connecting Y_{i-1} to Y_i abstracts a DAG with Y_{i-1} as a source and Y_i as a sink. Such a DAG has two possible forms according to the type of TE contained in an RRC (see below). Validating RRCs may result in blocking some resources—those for which the current time does not satisfy their associated TEs (refined by the RRC itself)—by resetting their associated clocks. The validation of several different RRCs could block the same resource r more than once by keeping on resetting $\mathtt{rY}$.

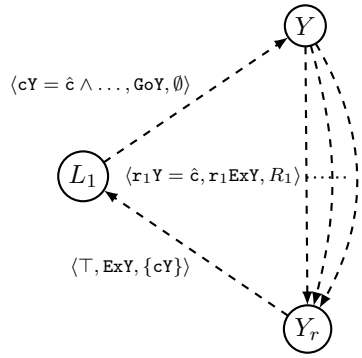
Finally, m transitions connect Y_n to Y_r, where m is the number of resources committable for Y. Taking one of these transitions means to commit one and only one *committable* resource among those associated to Y. Recall that, at any time, $r \in \mathcal{R}(Y)$ is committable if and only if $\mathtt{rY} = \hat{c}$. Therefore, none, a few or all of these transitions could be disabled as their guards might be false.

If r is not committable, it means that r's current associated TE is violated. Since in the previous locations $Y_1, \ldots, Y_n$ all RRCs having Y as a target have been validated, at least one of these transitions must have blocked r by resetting $\mathtt{rY}$. If the run can enter Y_r, it means that at least one resource for Y is committable (i.e., survived the blocking process). Again, none, a few or all resources can survive this process. Each one of these transitions verifies that r_i is committable (having $\mathtt{r}_i\mathtt{Y} = \hat{c}$ as the unique clock constraint in its guard) and resets all clocks $\mathtt{r}_j\mathtt{Y}$ associated to all other resources $r_j \neq r_i$ committable for the same time point. The last (single) transition connecting Y_r to L_1 fixes the execution time of Y by resetting $\mathtt{cY}$. Eventually, if r is committed for Y at time $\hat{c} - \mathtt{cY}$, the following three conditions hold:

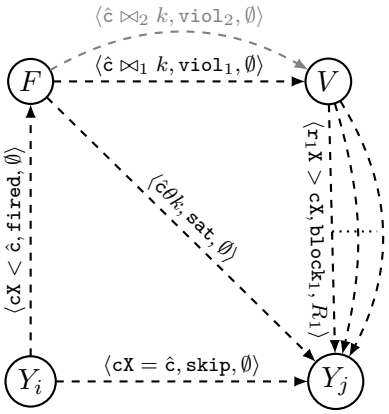
1. $r \in \mathcal{R}(Y)$



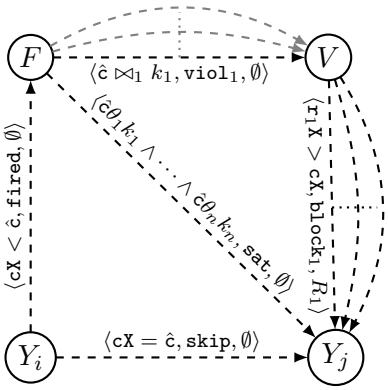
(a) Circular path modeling the execution of Y. A run enters this circular path only if Y is unexecuted (L_1 -►Y). Then, it verifies all the RRCs targeting Y (if any) (Y -► ··· -►Y_n). After that, it commits one and only one committable resource among those associated to Y (if any) (Y_n -►Y_r). Finally, it fixes the execution time of Y going back to L_1 (Y_r -►L_1). Thick edges labeled by "Validate RRC r_i" are a compact representation of the DAGs depicted in Figure 7(c)-(d).



(b) Circular path modeling the general execution of Y when no RRC targets Y. A run just commits one and only one committable resource among those associated to X and fixes the time of such an execution.



(c) Encoding RRCs specifying a single TE of Type 1 $\langle X, \theta k, Y, \rho \rangle$. A run enters F if only if the RRC has fired (Y_i -►F), otherwise it skips the verification (Y_i -►Y_j). At F, either θk is satisfied, and then no resource is blocked (F -►Y_j), or θk is violated (F -►V), and a few resources may eventually be blocked (V -►Y_j).



(d) Encoding RRCs specifying conjunctions of TEs of Type 1 $\langle X, \bigwedge_i \theta_i k_i, Y, \rho \rangle$. `skip`, `fired` and $\mathtt{block}_i$ transitions remain the same of those given in Figure 7c. `sat` extends by checking that the entire conjunction is true. Finally, there are as many $\mathtt{viol}_i$ transitions as the number of disjuncts arising from $\neg(\bigwedge_i \hat{\mathtt{c}}\theta_i k_i)$.

Figure 7: Modeling resource commitments: (a) shows the general circular path, (b) shows the case in which a time point does not appear as a target in any RRC, (c) and (d) give the encodings for Type 1 TEs and conjunctions of Type 1 TEs, respectively.

2. $\hat{\mathtt{c}} - \mathtt{cY} \models RE(r, Y)$
3. $\mathtt{rY} > \mathtt{cY}$ and $\mathtt{r}_i\mathtt{Y} = \mathtt{cY}$ for all $r_i \in \mathcal{R}(Y)$ such that $r \neq r_i$.

As a simple case, let us assume that a partial schedule for the considered CSTNUR executes only time point Y because some temporal constraints do not allow any further time point execution. In the corresponding TGA, after reaching L_1, no other urgent transition for executing another time point can be taken, because it is straightforward to show that the guards on other urgent transitions for executing other time points are false (see Section 3.2). Therefore, the TGA goes to L_0, where the properties presented in Section 5.1 hold:

- $L = L_0$, $\mathtt{c}_\delta = 0$ because transition `pass` resets it, $last(Executed) = \hat{\mathtt{c}}$ because L_1 is urgent.
- For the executed time point Y, $time(Y) = \hat{\mathtt{c}} - \mathtt{cY}$ and $res(Y) = \mathtt{rY}$, where $\mathtt{rY}$ is the only $\mathtt{r}_i\mathtt{Y}$ clock not reset as shown before.
- For each unexecuted time point X, $time(X) = \hat{\mathtt{c}}$ and $res(X)$ is not defined (i.e., all $\mathtt{r}_i\mathtt{X} = \hat{\mathtt{c}}$).

We now discuss in more detail how we encode RRCs and block resources. The simplest case is when no RRC targets a time point Y (Figure 7b). In such a case the circular path reduces to three locations only: L_1, Y and Y_r. Again, a run enters Y if the corresponding time point has not been executed yet, and then moves (for sure) to Y_r since no resource has been blocked. Finally, it fixes the execution time of Y.

Instead, when a time point Y appears as target in some RRC, we must make sure that, if this RRC has fired and the related TEs of the resources committable for Y have been updated, the global time $\hat{\mathtt{c}}$ must satisfy at least one of the TEs associated to different resources. This way, at least one resource can be committed to execute the time point.

In the rest of the paper we will only consider RRCs whose embedded TEs are either conjunctions of Type 1 TEs or of Type 2. We do not consider Type 0 TEs as current time (no matter its value) always satisfies them. Each Type 3 TE containing conjuncts of Type 1/Type 2 is transformed into a set of RRCs each containing either a Type 2 TE or a conjunction of Type 1 TEs. This can be done in linear time with respect to the number of conjuncts. For example, the RRC

$$\langle X, \underbrace{> 3 \wedge \leq 7}_{1} \wedge \underbrace{> C}_{2} \wedge \underbrace{\leq Z + 5}_{3}, Y, \rho \rangle$$

is composed by a conjunction of two Type 1 TEs (group 1) and two Type 2 TEs (groups 2 and 3). It is normalized as

$$\{\langle X, > 3 \wedge \leq 10, Y, \rho\rangle, \langle X, > C, Y, \rho\rangle, \langle X, \leq Z + 5, Y, \rho\rangle\}$$

We now proceed by discussing how we encode RRCs in the circular path. As we discussed at the beginning of this section, we validate the j^{th} RRC r_j by going from Y_i to Y_j. There are two possible cases: (1) the considered RRC contains a conjunction of TEs of Type 1, or (2) the considered RRC contains a TE of Type 2.

5.2.1 Encoding RRCs containing (a conjunction of) TEs of Type 1

We encode an RRC r_j having the form $\langle X, \theta k, Y, \rho \rangle$ by means of four locations Y_i, Y_j, F, and V (see Figure 7c) and we connect such locations as discussed below.

Entering F means that the RRC has *fired* (i.e., that X has been executed), whereas entering V means that both the RRC has fired and the TE of Type 1 specified in it is *violated*. Therefore, starting from Y_i, there are two possible transitions: one going to F (`fired`) and one going to Y_j (`skip`).

At F we have two transitions: either $t \models \theta k$ or not.

In the first case, no resource will be blocked since current time satisfies the TE. In such a case, the run moves to Y_j (`sat` transition) and goes ahead. The guard of `sat` contains the clock constraint $\hat{\mathtt{c}}\theta k$ modeling $t\theta k$. From Definition 16, it holds that $t \models \theta k$ iff $t\theta k$ whatever θ is. Since in a CSTNUR TEs are evaluated with respect to global time (modeled by $\hat{\mathtt{c}}$), it is sufficient to substitute $\hat{\mathtt{c}}$ for t getting $\hat{\mathtt{c}}\theta k$.

In the second case, $t \not\models \theta k$ and thus we must block all resources having θk associated. To achieve this aim, the run first enters V. Either one or two transitions connect F to V according to which θ is specified. If θ is '=' (i.e., the corresponding clock constraint is $\hat{\mathtt{c}} = k$), then there are two transitions, one having guard $\hat{\mathtt{c}} < k$ and the other having guard $\hat{\mathtt{c}} > k$. Since $\neg(\theta k)$ is true, then one of these two transitions must be true. If $\theta \in \{<, >, \leq, \geq\}$, then we just need to specify a unique transition going from F to V whose guard is $\neg(\hat{\mathtt{c}}\theta k)$. In Figure 7c, such transitions have labels $\mathtt{viol}_1$ and $\mathtt{viol}_2$, where $\mathtt{viol}_2$ (drawn in gray) exists if and only if θ is '='. $\bowtie_1$ and $\bowtie_2$ model the new θ operators arising from the negation of θk. Finally, a set of transitions connects V to Y_i. There exists one transition for each resource $r_X \in \mathcal{R}(X)$ saying that if r_X was committed for X, then all resources r_Y associated to Y such that the pair (r_X, r_Y) belongs to the relation ρ expressed in the considered RRC must be blocked by resetting their associated clocks. That is, the guard of each transition $\mathtt{block}_i$ is $\mathtt{r}_i\mathtt{X} > \mathtt{cX}$ (i.e., r_i was committed for X), whereas each update specifies the set R_i of clocks to reset. Each R_i is computed as follows:

$$R_i = \{\mathtt{r}_j\mathtt{Y} \mid r_j \in \mathcal{R}(Y) \wedge (r_i, r_j) \in \rho\}$$

We do not leave "anything behind" as all of these transitions are mutually-exclusive.

Now that we have discussed how to encode an RRC containing a single TE of Type 1, we consider RRCs containing TEs of Type 3 where each conjunct is of Type 1, i.e., RRCs having the form $\langle X, \bigwedge_i^n \theta_i k_i, Y, \rho \rangle$. Figure 7d shows the encoding of such an RRC, where rather than encoding $\langle X, \bigwedge_i^n \theta_i k_i, Y, \rho \rangle$ and obtaining $\langle X, \theta_1 k_1, Y, \rho \rangle$, ..., $\langle X, \theta_n k_n, Y, \rho \rangle$ and then encoding each RRC according to Figure 7c resulting in a circular path $Y_1 \dashrightarrow \cdots \dashrightarrow Y_n \dashrightarrow Y_r$ of n DAGs, we generate a refined shorter path $Y_1 \dashrightarrow Y_r$, consisting of one DAG only, ables to deal with the entire conjunction of TEs. This encoding substantially extends the `sat` and $\mathtt{viol}_i$ transitions in Figure 7c. All other transitions remain the same. We refine `sat` so that it verifies the clock constraint $\hat{\mathtt{c}}\theta_1 k_1 \wedge \cdots \wedge \hat{\mathtt{c}}\theta_n k_n$. To generate $\mathtt{viol}_i$ transitions, we proceed exactly as we did in Figure 7c. That is, we compute $\neg(\hat{\mathtt{c}}\theta_1 k_1 \wedge \cdots \wedge \hat{\mathtt{c}}\theta_n k_n)$ resulting in $\bigvee_j \neg(\hat{\mathtt{c}}\theta_i k_j)$, where, again, if θ is '=' in some conjunct of the initial TE, we generate two disjuncts.

As an example, consider the following RRC having a single Type 1 TE: $\langle X, =6, Y, \rho\rangle$. TE $=6$ is translated as clock constraint $\hat{c} = 6$ (`sat`), and $\neg(\hat{c} = 6)$ becomes $(\hat{c} < 6) \vee (\hat{c} > 6)$, from which $\texttt{viol}_1$, and $\texttt{viol}_2$ are generated (connecting F to V).

Let us now consider an RRC having a conjunction of Type 1 TEs: $\langle X, \leq 7 \wedge > 5, Y, \rho\rangle$. It follows that, $\leq 7 \wedge > 5$ becomes the clock constraint $\hat{c} \leq 7 \wedge \hat{c} > 5$ (`sat`), and $\neg(\hat{c} \leq 7 \wedge \hat{c} > 5)$ becomes $(\hat{c} > 7) \vee (\hat{c} \leq 5)$, from which we generate $\texttt{viol}_1$ and $\texttt{viol}_2$ connecting F to V.

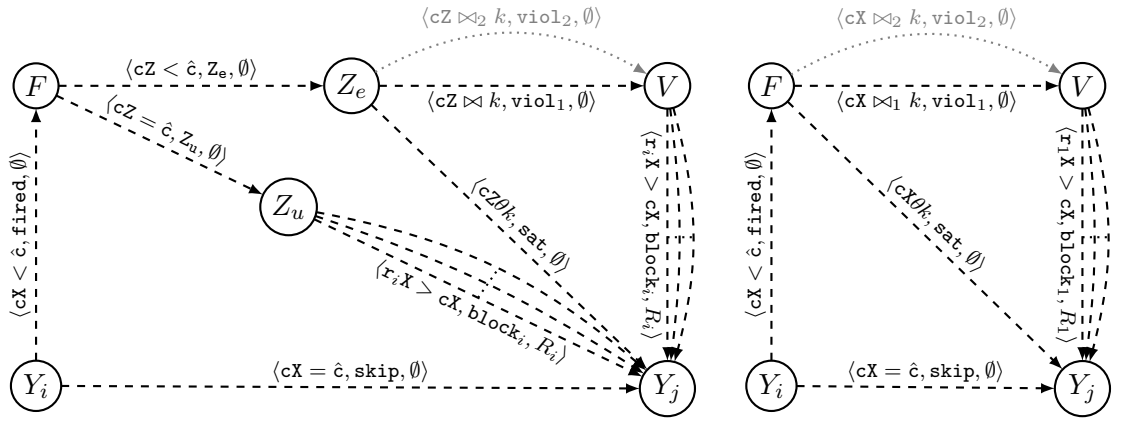
5.2.2 Encoding an RRC containing a TE of Type 2

We encode an RRC having the form $\langle X, \theta Z + k, Y, \rho\rangle$ by building a DAG consisting of the locations Y_i, Y_j, F, V, Z_e and Z_u (see Figure 8a) and connecting such locations as discussed below. The meaning of F and V as well as that of the transitions from Y_i to F and to Y_j are the same of those given for RRCs expressing TEs of Type 1 (Figure 7c). At F, the RRC has fired and the TEs related to some resources associated to Y have been updated (possibly differently depending on whether or not Z has been executed). Indeed, if Z has been executed, then $\theta Z + k$ is equivalent to $\theta(t_Z + k)$, where t_Z is the time at which Z has been executed. If Z has not yet been executed, t_Z is assumed to be $+\infty$ and thus $\theta Z + k$ is either $\top$ or $\bot$ depending on θ. Therefore, instead of connecting F to both V and Y_j as we have done before, we connect F to Z_e and Z_u, modeling the fact that Z has, or has not, been executed, respectively. A run moves to Z_e if and only if Z has been executed ($\texttt{cZ} < \hat{c}$), whereas it moves to Z_u if and only if Z has not ($\texttt{cZ} = \hat{c}$).

At Z_u we might block some resource(s) according to θ. If $\theta \in \{=, >, \geq\}$, then $t \not\models \theta Z + k$. Therefore, from Z_u to Y_j, there are as many blocking transitions as the number of resources in $\mathcal{R}(X)$. Each one specifies the set R_i as we have done for RRCs having TEs of Type 1. If $\theta \in \{<, \leq\}$, then the run moves to Y_j not blocking any resource (indeed, $t \models \theta Z + k$ implies $R_i = \emptyset$).

At Z_e, we must valuate $\theta Z + k$; thus, this TE becomes the clock constraint $\hat{c}\,\theta\,(\hat{c} - \texttt{cZ} + k)$, which simplifies to $\texttt{cZ}\,\theta\,k$. If $\texttt{cZ}\,\theta\,k$ is true, no resource will be blocked and a `sat` transition allows the run to move to Y_j. If $\texttt{cZ}\,\theta\,k$ does not hold, then $\neg(\texttt{cZ}\,\theta\,k) = \texttt{cZ} \bowtie k$ and, therefore, there are one or two $\texttt{viol}_i$ transitions allowing the run to move to V. At V the run moves to Y_j by (possibly) blocking some resources for Y and generating again a blocking transition for each resource associated to the firing time point X as we did for RRCs having TEs of Type 1.

Now that we have discussed the general encoding for RRCs whose TEs are of Type 2, we consider those RRCs where the firing time point and the time point belonging to the embedded TE are the same, i.e., those having form $\langle X, \theta X + k, \rho, Y\rangle$. For this case, we can remove the redundancy of the original encoding given in Figure 8a (locations Z_e and Z_u). Figure 8b shows such an encoding. Differently from what we discussed for the case depicted in Figure 7b, this encoding does not result in reducing the number of DAGs, but it avoids generating Z_e and Z_u for each DAG going from Y_{i-1} to Y_i. Indeed, when $X = Z$, keeping Z_e and Z_u would correspond to checking twice if X has been executed or not. Therefore, this encoding first removes Z_e and Z_u along with the transitions to enter these locations and, then, it connects F directly to Y_j and to V, maintaining the same `sat` and $\texttt{viol}_{1,2}$ transitions.



(a) Encoding normalized RRCs whose TEs are of Type 2 $\langle X, \theta Z + k, Y, \rho\rangle$. A run enters F if only if the RRC has fired (Y_i- ➝F), otherwise it skips the verification (Y_i- ➝Y_j. At F, either Z has been executed (F- ➝Z_e) or not (F- ➝Z_u). At Z_u a few resources might eventually be blocked (Z_u- ➝Y_j). At Z_e either $\theta Z + k$ is satisfied and no resource is blocked (Z_e- ➝Y_j), or $\theta Z + k$ is violated (Z_e- ➝V) and a few resources might be blocked (V- ➝Y_j).

(b) Optimizing the encoding of normalized RRCs whose TEs of Type 2 have the form $\langle X, \theta X + k, Y, \rho\rangle$. `skip`, `fired`, `sat`, `viol`$_i$ and `block`$_i$ transitions remain the same of those given in Figure 8a (they just connect different locations).

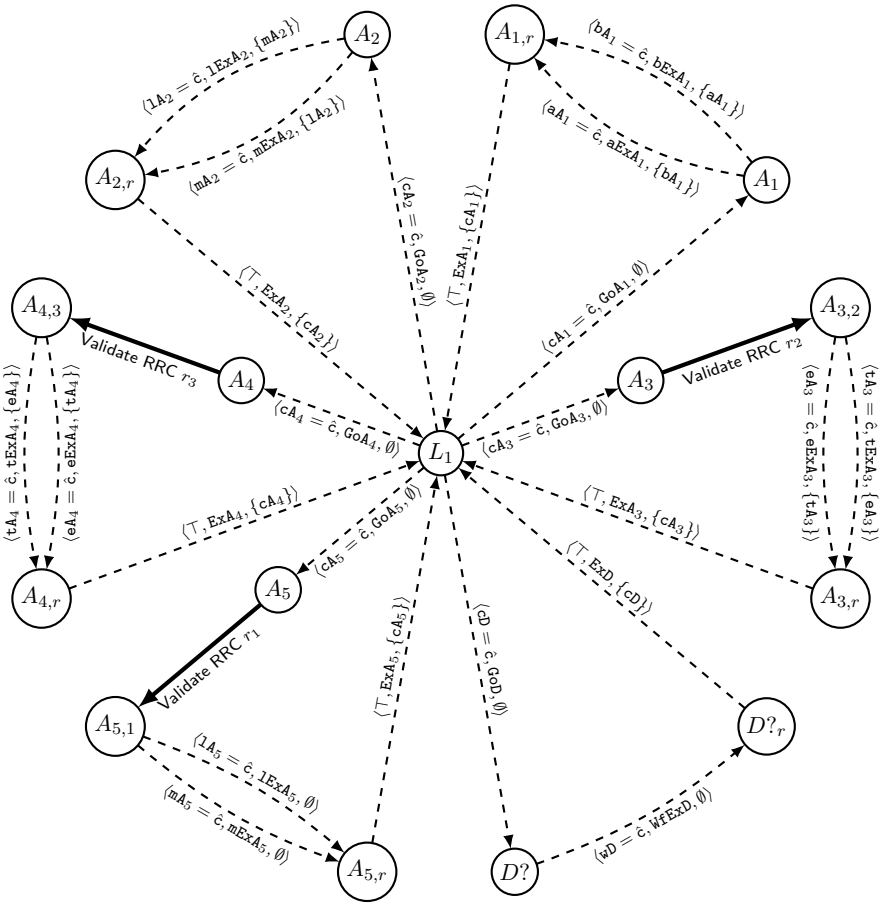
Figure 8: Encoding RRCs specifying TEs of Type 2 $\langle X, \theta Z + k, Y, \rho\rangle$: (a) shows the general encoding, and (b) refines it for the case $X = Z$.

Figure 9a shows the circular paths modeling the resource commitments for the execution the non-contingent time points $D?$, A_1, A_2, A_3, A_4 and A_5 of the CSTNUR in Figure 6, whereas Figure 9b and Figure 9c detail the validation of the related RRCs (thick edges in Figure 9a) for the circular paths modeling the authorized execution of A_3, A_5, respectively. We do not discuss the encoding for A_4 as it is similar to that for A_3.

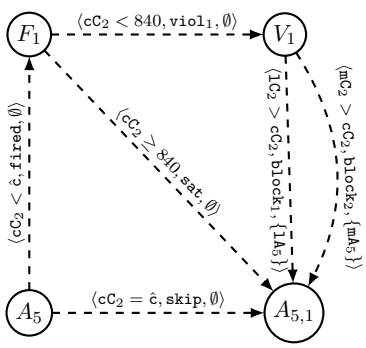
5.3 Encoding Contingent Time Points into Contingent Circular Paths

In the encoding for CSTNUs, transitions modeling the execution of the contingent time points are *controllable* self-loop transitions at L_0. In the semantics we gave for CSTNURs (see Section 4.2), the two time points of a contingent link (A, x, y, C) must be executed by committing the same resource. Moreover, contingent time points cannot appear as targets in any RRC. Therefore, for each contingent time point the encoding generates a *contingent circular path* similar to that depicted in Figure 7b neglecting the validation of RRCs.

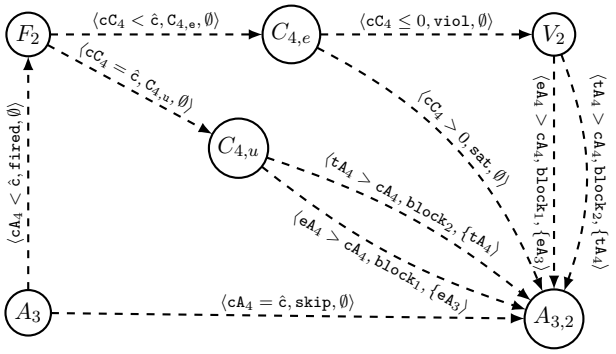
The path starts and ends at L_0 and contains two internal *urgent* locations C and C_r (see Figure 10a). A unique transition `GoC` goes from L_0 to C, whereas a set of $|\mathcal{R}(A)|$ transitions connects C to C_r. A run can enter C as soon as the clock constraint for contingent time points (i.e., $\mathtt{cA} < \hat{\mathtt{c}} \wedge \mathtt{cC} = \hat{\mathtt{c}} \wedge \mathtt{cA} \geq x \wedge \mathtt{cA} \leq y$) becomes true. `GoC` is unique and does not reset any clock. After that, the resource that was committed for A is committed for C as well by means of a transition $\mathtt{r}_i\mathtt{ExC}$ going from C to C_r. The only enabled transition $\mathtt{r}_i\mathtt{ExC}$ is the one whose guard contains the clock constraint $\mathtt{r}_i\mathtt{C} > \mathtt{cA}$, i.e., the transition associated to the resource r_i committed for A. Indeed, for all other $r_j \in \mathcal{R}(A)$ where $r_j \neq r_i$, it holds



(a) Circular paths for assigning a value and a resource to time points A_1, A_2, A_3, A_4, A_5, and $D?$. For the sake of simplicity, the guard of each transition from L_1 to A_1, A_2, A_3, A_4, A_5, and $D?$ does not contain the extended parts introduced in Section 3.2.

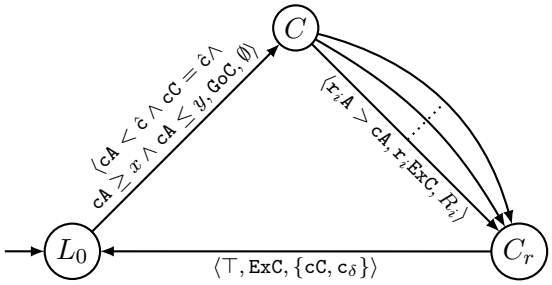


(b) Validation of RRC $r_1 = \langle C_2, \geq C_2 + 840, A_5, = \rangle$.

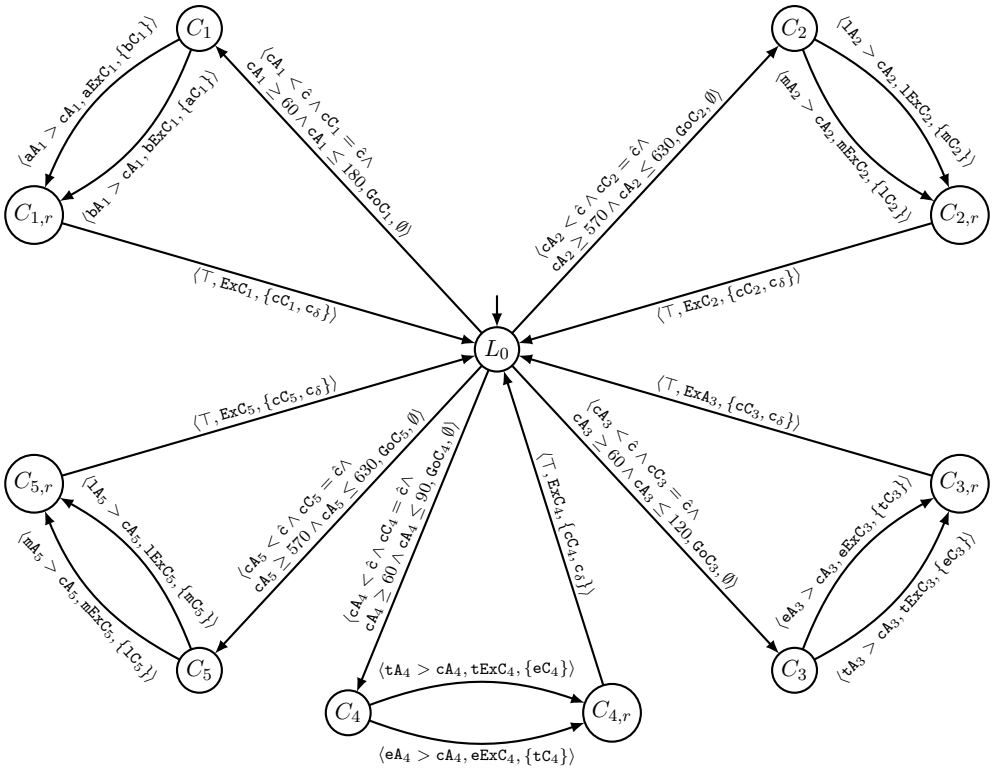


(c) Validation of RRC $r_2 = \langle A_4, > C_4, A_3, \neq \rangle$.

Figure 9: Circular paths modeling the authorized execution of the non-contingent time points of the CSTNUR in Figure 6 and two RRC encodings.



(a) Modeling the authorized execution of a generic contingent time point C. This path just commits the same resource r_i that was committed for the corresponding activation time point A. Once A has been executed, only one transition connecting $C \longrightarrow C_r$ will be enabled. For each of these transitions $R_i = \{\mathtt{r}_j\mathtt{C} \mid r_j \in \mathcal{R}(C) \wedge r_j \neq r_i\}$.



(b) The encoding of C_1, C_2, C_3, C_4, and C_5 of Figure 6.

Figure 10: Modeling the executions of contingent time points: (a) shows the pattern for modeling the execution of a contingent time point C, whereas (b) shows the encoding of C_1, C_2, C_3, C_4, and C_5 of Figure 6.

that $\mathtt{r}_j\mathtt{C} \leq \mathtt{cA}$. Finally, the run moves back to L_0 by resetting $\mathtt{cC}$, i.e., fixing the execution time for C and $\mathtt{c}_\delta$.

Even in this case, it is straightforward to show that the properties presented in Section 5.1 hold when the TGA returns to state L_0.

Figure 10 sums up the general pattern for modeling the execution of contingent time points (Figure 10a) and the application of such a pattern to the contingent time points of the CSTNUR in Figure 6 (Figure 10b).

6. Complexity and Correctness of the Encoding

In this section, we discuss the computational temporal complexity and the correctness of encoding a CSTNUR $\mathcal{N}$ into a TGA. In Section 5 we introduced in a constructive way how to rewrite RRCs in the corresponding DAGs. Therefore, firstly, we discuss the temporal computational-complexity, hereinafter complexity, of encoding each single RRC. Then, we discuss the complexity to connect all these encodings into the TGA and, finally, the complexity to encoding the check of all temporal constraint into the winning path. The overall complexity is the sum of all these steps and it will result to be polynomial with respect to the size of the network.

Lemma 1 (RRC Encoding Complexity)**.**
Given a CSTNUR $\mathcal{N} = \langle \mathcal{T}, \mathcal{P}, L, \mathcal{OT}, O, \mathcal{C}, \mathcal{L}, \mathcal{R}, RA, RE, \mathcal{RRC} \rangle$, for each RRC $r \in \mathcal{RRC}$, the complexity for determining its corresponding TGA fragment (see Section 5) is $\mathcal{O}(|\mathcal{R}|^2)$, where $\mathcal{R}$ is the set of resources.

Proof. Let us consider the patterns for building TGA fragments from RRCs depicted in Figure 7c (for RRC having only Type 1 TE), Figure 7d (for RRC having conjunction of Type TEs), and Figure 8a (for RRC having a Type 2 TE). For each pattern, the number of TGA nodes is 6 at most and the number of edges representing block actions is equal to the possible number of resources that can be associated to a CSTNUR node. In the worst case, this number of resources is equal to $|\mathcal{R}|$. Moreover, each TGA block edge $\texttt{block}_i$ has to reset clocks associated to some resources. Indeed, in the worst case, its label may have length $\mathcal{O}(|\mathcal{R}|)$ as it must contain the references to clocks associated to all the other resources. Therefore, in the worst case, the complexity of building a TGA fragment associated to a RRC is $\mathcal{O}(|\mathcal{R}|^2)$. □

Lemma 2 (Resource Commitments Encoding Complexity)**.**
Given a CSTNUR $\mathcal{N} = \langle \mathcal{T}, \mathcal{P}, L, \mathcal{OT}, O, \mathcal{C}, \mathcal{L}, \mathcal{R}, RA, RE, \mathcal{RRC} \rangle$, the complexity for determining all TGA fragments corresponding to the resource commitments (as, for example, all fragments in Figure 9a) is $\mathcal{O}(n^3)$, where n is the encoding length of $\mathcal{N}$.

Proof. For each non-contingent time point Y, the modeling of its execution and its resource commitment is given by a TGA circular path $L_1 \dashrightarrow Y \dashrightarrow Y_1 \dashrightarrow \cdots \dashrightarrow Y_n \dashrightarrow Y_r \dashrightarrow L_1$ (cf. Figure 7a). In such path, the more expensive fragments to build are:

- the edge $L_1 \dashrightarrow Y$, because it represents the TGA action for evaluating whether Y is not yet executed and whether its (possible) predecessors have been already executed. The label of this edge may contain $\mathcal{O}(|\mathcal{P}|)$ clocks specifications, where $\mathcal{P}$ is the set of possible propositions of the network. Therefore, its building complexity is $\mathcal{O}(|\mathcal{P}|)$.
- each fragment $Y_{i-1} \dashrightarrow Y_i$, because it represents a DAG handling the validation of the i^{th} RRC associated to Y. The complexity of each fragment is $\mathcal{O}(|\mathcal{R}|^2)$ as proved in Lemma 1, and

- the edges $\mathtt{r}_i\mathtt{ExY}, i = 1, \ldots, m$ from Y_n to Y_r, because they can be $|\mathcal{R}|$ at most and each of them can have label with length $\mathcal{O}(|\mathcal{R}|)$. Indeed, in the worst case, each action (i.e., edge) is associated to a resource and it has to reset all clocks associated to all the other resources. Therefore, the complexity for building such edges is $\mathcal{O}(|\mathcal{R}|^2)$.

For each non-contingent time point, the complexity for building the corresponding circular path without RRCs is $\mathcal{O}(|\mathcal{P}| + |\mathcal{R}|^2)$. The complexity for building all RRC fragments is $\mathcal{O}(|\mathcal{RRC}||\mathcal{R}|^2)$, where $\mathcal{RRC}$ is the set of all RRCs in the considered CSTNUR. The cost for positioning all RRC fragments into the right circular paths is $\mathcal{O}(|\mathcal{T}||\mathcal{RRC}|)$. Therefore, the complexity for representing into a TGA the execution of all non-contingent time points with their resource commitments is $\mathcal{O}(|\mathcal{T}|(|\mathcal{P}|+|\mathcal{R}|^2+|\mathcal{RRC}|)+|\mathcal{RRC}||\mathcal{R}|^2) = \mathcal{O}(n^3)$, where n is the length of the $\mathcal{N}$ encoding. □

Lemma 3 (Contingent Time Point Encoding Complexity)**.**
Given a CSTNUR $\mathcal{N} = \langle \mathcal{T}, \mathcal{P}, L, \mathcal{OT}, O, \mathcal{C}, \mathcal{L}, \mathcal{R}, RA, RE, \mathcal{RRC} \rangle$, the complexity for determining all TGA fragments for modeling the execution of contingent time points (cf. Figure 10a) is $\mathcal{O}(n^3)$, where n is the encoding length of $\mathcal{N}$.

Proof. For each contingent time point C, the modeling of its execution is given by a TGA circular path $L_0 \rightarrow C \rightarrow C_r \rightarrow L_0$ (cf. Figure 10a), where between C are C_r there are $|\mathcal{R}|$ edges at most. Each of the edges connecting C to C_r commits a resource for executing C resetting the clocks associated to all the other resources. Therefore, in the worst case, each label of such edges may contain $|\mathcal{R}| - 1$ clock resets.

As a consequence, for each contingent time point, the complexity for building the corresponding TGA fragment is $\mathcal{O}(|\mathcal{R}|^2)$ and the overall complexity for building all TGA fragments is $\mathcal{O}(n^3)$, where n is the encoding length of $\mathcal{N}$. □

Theorem 1. *Any CSTNUR $\mathcal{N} = \langle \mathcal{T}, \mathcal{P}, L, \mathcal{OT}, O, \mathcal{C}, \mathcal{L}, \mathcal{R}, RA, RE, \mathcal{RRC} \rangle$ can be encoded into a corresponding TGA in $\mathcal{O}(n^3)$ time, where n is the encoding length of $\mathcal{N}$.*

Proof. In Section 5 we presented how to encode CSTNURs into TGAs by extending the encoding presented in Section 3.2. In particular, in Section 5 we propose to replace all TGA transitions encoding the execution of non-contingent and contingent time points with circular paths presented in Section 3.2 while maintaining the encoding of winning path for checking that all relevant temporal constraints are satisfied.

By Lemma 2 and Lemma 3, the complexity for building all circular paths in the TGA is $\mathcal{O}(n^3)$, where n is the encoding length of $\mathcal{N}$.

The complexity for encoding the winning path is linear in the number of distinct labels present in the input network (Cimatti et al., 2016)). The number of distinct labels in $\mathcal{N}$ is $\mathcal{O}(n)$.

Therefore, the overall complexity for building a TGA from a CSTNUR $\mathcal{N}$ is $\mathcal{O}(n^3)$. □

We prove the correctness of the encoding given in Section 5 by means of the following two theorems. Such theorems extend those given in (Cimatti et al., 2016) proving the correctness of the CSTNU-to-TGA encoding. Our extension takes into account resources and RRCs. We first prove the equivalence between the execution semantics of the resulting TGA and the semantics of RTEDs (Theorem 2), and then that any counter-strategy for `ctrl`

synthesized by reachability analysis of the resulting TGA corresponds to an RTED-based strategy (Theorem 3).

Theorem 2. *Let $\mathcal{N} = \langle \mathcal{T}, \mathcal{P}, L, \mathcal{OT}, O, \mathcal{C}, \mathcal{L}, \mathcal{R}, RA, RE, \mathcal{RRC} \rangle$ be a CSTNUR and let $\mathcal{G}$ be the encoding of $\mathcal{N}$ into a TGA, as described in Section 5. Then $\mathcal{G}$ correctly captures the execution semantics for $\mathcal{N}$ in the sense that any sequence of partial schedules that can be generated for $\mathcal{N}$ according to the execution semantics for CSTNURs corresponds to a run for $\mathcal{G}$ that can be generated by following its transitions according to the TGA semantics.*

Proof. We show that any sequence of partial schedules that can be generated for any CSTNUR according to the execution semantics given in Section 4.2 corresponds to a run for the equivalent TGA that can be generated by following its transitions according to the classic TGA semantics.

In the following, we write $\vec{c} = k$ meaning that all clocks in $\vec{c}$ are equal to k and we write $\vec{c} + k$ meaning that we refer to value of clocks in $\vec{c}$ increased of k each.

The proof is given by induction considering the already introduced invariants, recalled here for sake of readability.

Invariant Each locally consistent partial schedule that can be generated for $\mathcal{N}$ corresponds to a state $(L, \vec{c})$ of the TGA $\mathcal{G}$ in which:

- $L = L_0$, $\mathtt{c}_\delta = 0$, $last(Executed) = \hat{\mathtt{c}}$.
- For each executed time point X, $time(X) = \hat{\mathtt{c}} - \mathtt{cX}$ and $res(X) = \mathtt{rX}$, where $\mathtt{rX}$ is the only $\mathtt{r}_i\mathtt{X}$ clock not reset.
- For each unexecuted time point X, $time(X) = \hat{\mathtt{c}}$ and $res(X)$ is not defined (i.e., all $\mathtt{r}_i\mathtt{X} = \hat{\mathtt{c}}$).
- For each executed observation time point $P?$, if $p = \top$ then $\mathtt{bP} = \hat{\mathtt{c}}$, and if $p = \bot$ then $\mathtt{bP} < \hat{\mathtt{c}}$. If the time point is not executed, the value of $\mathtt{bP}$ means nothing.

Base case. The initial $\mathcal{PS}$ corresponds to the initial state $(L_0, \vec{c})$ where $\vec{c} = 0$. This partial schedule is trivially locally consistent.

Inductive step. Suppose that $\mathcal{PS}$ is a locally consistent partial schedule that has been generated according to the execution semantics for CSTNURs, and that $\mathcal{PS}$ satisfies the above invariant. Let $(L_i, \vec{c}_i)$ be the corresponding state of the TGA. Since $\mathtt{c}_\delta = 0$, the only transitions that are immediately enabled are those entering the contingent circular paths and those that set truth values to propositions. In case a run enters a contingent circular path corresponding to the executing of a contingent time point C, it enters location C and then it must move to location C_r, representing the event that a resource has been committed, picking the transition having the same resource that was committed for the related activation time point A. Finally, it must move back to L_0. Since the homonymous location C and the location C_r are urgent, time does not elapse. All transitions executed during a walk through contingent circular paths and the transitions modeling the truth value assignments represent the instantaneous reactions of `env`, in which a set of one or more contingent time points and/or proposition assignments can be executed simultaneously. Suppose that `env` does not take any transition when $\mathtt{c}_\delta = 0$. As soon as $\mathtt{c}_\delta > 0$, both `ctrl`

and env may execute enabled transitions (i.e., those with true guards). For example, env might decide to execute one or more contingent time-points $C_1, \ldots, C_n$ when $\mathtt{c}_\delta = 3$. That would correspond to $\Delta_{\mathtt{env}} = (k, \{C_1, \ldots, C_n\})$, where $k = last(Executed) + 3$.

Each time env takes a transition pFalse to reset the clock associated to transition p (i.e.,, setting p to $\bot$) or a transition ExC to execute a contingent time point, $\mathtt{c}_\delta$ is reset to 0, making ctrl unable to interrupt env during the execution of the initiated transition. Thus, at these time instants, it holds that $\Delta_{\mathtt{ctrl}} = \mathtt{wait}$ and the resulting outcomes are exactly the ones described in cases 1 and 2 of Definition 22. The guard of the env 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 in an instant such that $C - A \in [x, y]$. Likewise, for a truth value assignment, the fail transition that ctrl can take (if $\delta > 0$) ensures that env assigns a truth value to a proposition instantaneously after the execution of the observation time point.

Also, when env's sequence of "simultaneous" transitions completes, $\hat{\mathtt{c}}$ equals the time of the most recent execution (e.g., $last(Executed) + 3$). In addition, for each newly executed time-point C, the clock cC is reset and for each $r_i \in \mathcal{R}(C)$, if $\mathtt{r}_i\mathtt{C} = \hat{\mathtt{c}}$ then $\mathtt{r}_i\mathtt{A} = \hat{\mathtt{c}}$ and if $\mathtt{r}_i\mathtt{C} < \hat{\mathtt{c}}$ then $\mathtt{r}_i\mathtt{A} < \hat{\mathtt{c}}$ ensuring that $\hat{\mathtt{c}} - \mathtt{cC}$ equals the execution time of C. Analogously, for each pFalse taken, it holds that $\mathtt{bP} < \hat{\mathtt{c}}$. Since cC is reset only once and each proposition is assigned only once, the values of $\hat{\mathtt{c}} - \mathtt{cC}$ and bP remain fixed forever.

Instead, suppose that ctrl has decided to commit a set of resources to execute a set of non-contingent time points before env executes some contingent time points (for example, ctrl has decided to execute some time points at time $last(Executed) + 2$). This situation results in ctrl taking the gain transition to take back control and then, once in its location, instantaneously go through the circular paths (for non-contingent time points) to commit the resources to execute those time points at that time, and immediately returning to the env location by means of the pass transition. Since all the locations but L_0 are urgent, $\hat{\mathtt{c}}$ does not change its value (in our example, it is $last(Executed) + 2$). The sequence of transitions to go through the circular paths corresponds to the partial outcome in Definition 22 (cases 3 and 4) where $\Delta_{\mathtt{ctrl}} = (t, \{(r_1, X_1), \ldots, (r_n, X_n)\})$, $t = last(Executed) + 2$, and for each $(r, X) \in NonContingent$ (of $\Delta_{\mathtt{ctrl}}$), $r \in \mathcal{R}(X)$.

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

Theorem 3. *Let $\mathcal{N}$ be a CSTNUR, $\mathcal{G}$ be the encoding of $\mathcal{N}$, and $\sigma_{\mathcal{G}}$ be a winning TGA counter-strategy for* ctrl*. Then, there is an equivalent RTED-based strategy $\sigma_{\mathtt{ctrl}}$ for* ctrl *that will ensure the satisfaction of all temporal constraints in $\mathcal{N}$ and all RRCs, if fired, whatever the contingent durations and truth value assignments.*

Proof. If $\mathcal{N}$, $\mathcal{G}$, $\sigma_{\mathcal{G}}$ are as assumed, then we can interpret $\sigma_{\mathcal{G}}$ as a function $\sigma_{\mathcal{G}} \colon S \to Act \cup \mathtt{wait}$, where S is the state set of the TGA and Act is the ctrl action set (equivalently, the set of uncontrollable transitions).

Suppose the TGA has just got into the state $(L_0, \vec{c})$. As we have already noted, for any time point X associated to clock $\mathtt{cX}$, it holds that:

- $\mathtt{cX} = \hat{\mathtt{c}}$,
- all $\mathtt{r}_i\mathtt{X}$ have value $\hat{\mathtt{c}}$ before X executes, and
- all $\mathtt{r}_i\mathtt{X}$ but one have value $\mathtt{cX} < \hat{\mathtt{c}}$ after X executes. The clock $\mathtt{r}_i\mathtt{X}$ remaining equal to $\hat{\mathtt{c}}$ represents that the corresponding resource has been committed to execute X.

For each observation time point $P?$, when $\mathtt{cP} = \hat{\mathtt{c}}$, the associated proposition modeled by $\mathtt{bP}$ is $\mathtt{bP} = \hat{\mathtt{c}}$ (i.e., unknown before $P?$ executes), and when $\mathtt{cP} < \hat{\mathtt{c}}$, either $\mathtt{bP} = \hat{\mathtt{c}}$ (i.e., $\top$) or $\mathtt{bP} < \hat{\mathtt{c}}$ (i.e., $\bot$) (i.e., $P?$ has been executed and its proposition is known). Thus, $(L_0, \vec{c})$ specifies a partial schedule.

Now, suppose that $\hat{\mathtt{c}} > \mathit{last}(\mathit{Executed})$, 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` and `ctrl` locations. In such a loop, `ctrl` has not executed any non-contingent time point, and `env` has just waited. Let $(L_0, \vec{c}_\prime)$ be a state preceding such loop. Then, for some $\epsilon > 0$, all the clocks in $\vec{c}$ equal those in $\vec{c}_\prime + \epsilon$, and by construction, $\mathit{last}(\mathit{Executed})$ refers to the clocks in $\vec{c}_\prime$. Next, let $d = \min\{d \mid \sigma_{\mathcal{G}}(L_0, \vec{c}_\prime + d) \neq \mathtt{wait} \wedge \sigma_{\mathcal{G}}(L_0, \vec{c}_\prime + d) \neq \mathtt{pass}\}$ be the minimum time that can elapse from $\vec{c}_\prime$ before the strategy $\sigma_{\mathcal{G}}$ recommends a transition different from `gain` and `pass`, and let $\vec{c}_0 = \vec{c}_\prime + d$. The unique sequence of execution transitions at `ctrl` is $\sigma_{\mathcal{G}}(L_1, \vec{c}_0), \ldots, \sigma_{\mathcal{G}}(L_1, \vec{c}_n)$, where each $\vec{c}_{i+1} = \vec{c}_i$, except for $\mathtt{cX}$ with X the time point executed by $\sigma_{\mathcal{G}}(L_1, \vec{c}_i)$. The termination of this sequence of transitions is guaranteed since time points are finite and can only be executed once. If $\sigma_{\mathcal{G}}(L_1, \vec{c}_n)$ is the last execution transition, then $\mathtt{pass} = \sigma_{\mathcal{G}}(L_1, \vec{c}_n)$. That transition leads back to the state $(L_0, \vec{c}_n)$, where $\vec{c}_n$ is the same as $\vec{c}_0$, except that the clocks for the time points executed by the transitions plus those for resources for those time points, $\sigma_{\mathcal{G}}(L_1, \vec{c}_0), \ldots, \sigma_{\mathcal{G}}(L_1, \vec{c}_n)$, are all 0 in $\vec{c}_n$.

Next, let t be the time at which $\sigma_{\mathcal{G}}$ recommends `ctrl` a non-trivial transition, and *NonContingent* be the set of pairs (resource, time point) corresponding to the execution transitions, $\sigma_{\mathcal{G}}(L_1, \vec{c}_0), \ldots, \sigma_{\mathcal{G}}(L_1, \vec{c}_n)$. Then $(t, \mathit{NonContingent})$ is a $\Delta_{\mathtt{ctrl}}$ corresponding to what the strategy recommends at $(L_0, \vec{c}_\prime)$. Note that `env` may decide to instantaneously react by executing some contingent point at time t too, an outcome that is prevented by the execution semantics for CSTNURs (Definition 22, cases 3 and 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 $\Delta_{\mathtt{ctrl}}$. Since the guard of `gain` requires a positive time delay, that $\Delta_{\mathtt{ctrl}}$ is properly prohibited from any kind of instantaneous reaction (by `ctrl`). This procedure gives a mapping from any $(L_0, \vec{c})$ state that is reachable following $\sigma_{\mathcal{G}}$. □

Finally, as proof-of-concept, we modeled and validated the motivating example for a round-trip flight process (see Section 2) encoding it into a corresponding TGA. The dynamic controllability of such an example has been determined by synthesizing a memoryless execution strategy using the well-known model checker Uppaal-TIGA (Behrmann, Cougnard,

David, Fleury, Larsen, & Lime, 2007). We verified that the example is dynamically controllable and that the model checking phase took 207 minutes and 28 seconds to synthesize a 1.6MB memoryless execution strategy as a *certificate of YES* for this decision problem. Appendix A presents more details about this proof-of-concept.

7. Related Work

In the following sections, we discuss the main contributions related to the uncertainty for conditions and for durations, and how these features have been combined also with access control policies and resource management.

7.1 Managing Decisions and Conditions

Temporal Plan Networks (TPNs) (Kim, Williams, & Abramson, 2001) extend STNs by adding decision nodes (as new labeled STN nodes) and symbolic constraints (as new labels for STN edges) to model temporal plans with controllable choices. Each outgoing edge of a decision node represents a decision. A symbolic constraint is either the label $\mathsf{Ask}(c)$ (is c true?) or the label $\mathsf{Tell}(c)$ (c is true!) where c is a literal. A TPN edge may be labeled with a symbolic constraint as well as a duration. Symbolic constraints may exclude nodes from being executed. A plan is consistent if it satisfies both temporal and symbolic constraints. TPNs do not specify more than one temporal constraint on the same edge. Consistency is checked by means of a backtracking algorithm that builds iteratively a sequence of decisions, (if any), for which all constraints are satisfied.

In (Tsamardinos, Vidal, & Pollack, 2003), the authors introduced Conditional Simple Temporal Networks even if they didn't name them as such. A *Conditional Simple Temporal Network (CSTN)* is a data structure for representing and reasoning about temporal constraints in domains where some constraints may apply only in certain scenarios. Each condition in a CSTN is represented by a propositional letter whose truth value is not controlled, but instead is observed in real time. An execution strategy for a CSTN specifies the times at which various time-points will be executed. Such a strategy can be dynamic in that its execution decisions can react to the information obtained from such observations. The *Conditional Simple Temporal Problem (CSTP)* is the problem of determining whether a given CSTN admits a dynamic execution strategy that can guarantee the satisfaction of all constraints no matter which combination of propositional outcomes happens to be observed over time. If such a strategy exists, the CSTN is said to be *dynamically consistent (DC)*. Thus, the CSTP is the DC-checking problem for CSTNs. The authors proposed to solve the CSTP by encoding it as a meta-level *Disjunctive Temporal Network (DTN)* (Stergiou & Koubarakis, 2000), where constraints may be specified as a set of disjunctive constraints between pairs of time points. Then, the corresponding DTN is solved by an off-the-shelf DTN solver. Although of theoretical interest, this approach is not practical because the CSTP-to-DTN encoding has exponential size and, on top of that, the DTN solver runs in exponential time.

In (Léauté & Williams, 2005), the authors provided a continuous model-based executive for systems having state variables and continuous dynamics. Their approach is based on encoding sub-parts of the main temporal problem into disjunctive linear programs (DLPs, (Balas, 1979)), reformulating them as Mixed-Integer Linear Programs, and solving these

last ones exploiting some constraint-pruning policies for both state constraints and temporal constraints. Their experimental evaluation shows that the adopted pruning policies enable the executive to design near-optimal control sequences in real time even if the worst case complexity of the problem is exponential.

Drake (Conrad & Williams, 2011) is an executive for temporal plans with choices modeled as *Labeled STNs*, which extend STNs by labeling constraints with environments (set of instantiated discrete decision variables). There are no decision points, and therefore decision variables can be set anytime. The main goal of the developers of Drake was to implement a dispatching executive with a lower memory footprint. To that end, Drake uses an Assumption-based Truth Maintenance System to maintain a minimal representation of the constraints needed to execute the network. During execution, decisions are discriminated by generating conflicts according to the time Drake decides to schedule some event. In (Conrad & Williams, 2011), the authors proved that Labeled STNs are equivalent to DTNs (Stergiou & Koubarakis, 2000). They also proved that, in general, a dispatchable solution found by Drake is more compact by over two orders of magnitude with respect to the equivalent one found with previous methods for DTNs.

In (Yu & Williams, 2013), the authors faced a slightly different problem: to face over-constrained temporal problems, they represent them as *Controllable Conditional Temporal Problems (CCTPs)* and solve them by the Best-first Conflict-Directed Relaxation (BCDR) algorithm. CCTP extends CSTP (Tsamardinos et al., 2003) assuming that all CSTP condition variables are controllable (in other words, they are decisions) and that domains of condition variables may be any finite domains instead of binary ones. Consequently, to solve a CCTP, it is sufficient to find one consistent set of discrete variable assignments. The BCDR algorithm enumerates the best continuous relaxations of a given network (CCTN) associated to a CCTP. In particular, the BCDR algorithm reformulates an over-constrained CCTN by identifying its continuously relaxable temporal constraints, whose bounds can be partially relaxed to restore consistency. It uses a conflict-directed strategy to enumerate continuous relaxations in best-first order: after learning conflicts between constraints and variable assignments, it uses the resolutions to these conflicts to guide the search away from infeasible regions. Since controllable choices are not dependent on observation events, solving CCTP is simpler than determining the dynamic/weak consistency of a CSTN.

Pike is an executive for temporal plans with both controllable and uncontrollable choices, i.e., conditions, that achieves plan recognition and adaptation concurrently (Levine & Williams, 2014). Pike employs *Temporal Plan Networks with (decision) Uncertainty (TPNUs)*, which extend TPNs to address both controllable and uncontrollable choices. Pike adapts controllable choices to uncontrollable ones made by the environment. As stated in (Levine & Williams, 2014), Pike takes as input (1) a TPNU, (2) the initial and goal states (sets of Planning Domain Definition Language (PDDL) predicates, (Fox & Long, 2003)), (3) a stream of state estimates (sets of predicates), and (4) a stream of time assignments and outcomes to the uncontrollable choices in the TPNU. Pike outputs (1) a stream of choice assignments to the TPNU's controllable variables, and (2) a dispatch of the TPNU's events, such that there is at least one complete and consistent candidate sub-plan for the choices made.

In (Hunsberger et al., 2015), the authors proposed *Conditional Simple Temporal Networks (CSTNs)* extending the CSTN implicitly proposed in (Tsamardinos et al., 2003) by

labeling both time points and constraints by conjunctions of propositional letters, i.e., conditions. In (Tsamardinos et al., 2003) only time points can be labeled. Each proposition is associated to an *observation time point*, a special type of time point that, when executed, allows the environment to set the proposition truth value. Dynamic consistency analysis ensures the existence of a strategy executing all (relevant) time points satisfying all (relevant) temporal constraints whatever the combination of proposition truth values revealed during the execution. The CSTP for these CSTNs is PSPACE-complete (Cairo & Rizzi, 2016).

In the previous models supporting the concept of condition, namely CSTN, CCTP, Pike and CSTN, is possible to represent uncertainty by means of a suitable use of conditions as shown, for example, by Hunsberger and Posenato (2018b). On the other side, CSTNURs cannot represent decisions as in TPNs, Drake and Pike. The CSTN (Tsamardinos et al., 2003; Hunsberger et al., 2015) models are subsumed by the CSTNUR one. Indeed, a CSTNUR instance without any resource and contingent link is a CSTN instance. Moreover, all the previous models do not consider and do not allow a compact representation and management of resources and of related constraints.

7.2 Managing Conditional and Temporal Uncertainty

A *Conditional Simple Temporal Network with Uncertainty (CSTNU)* merges the semantics of STNUs and CSTNs to deal with both conditional constraints and uncontrollable durations simultaneously (Hunsberger et al., 2012). Hunsberger and Posenato (2018a) proposed a sound-and-complete constraint-propagation-based algorithm for checking the dynamic controllability (DC) of a CSTNU that results to be more practical than the DC algorithms based on TGA proposed by Cimatti et al. (2016).

A *Controllable Conditional Temporal Problem with Uncertainty (CCTPU)* extends CCTP to address temporal plans with also uncertain durations (Yu, Fang, & Williams, 2014). A Conflict-Directed Relaxation with Uncertainty algorithm (CDRU) is provided to deal with over constrained temporal problems extending the result obtained for CCTPs. In particular, the CDRU algorithm generates continuous relaxations restoring both strong and dynamic controllability. It extends the conflict learning and resolution process in previous relaxation algorithms to account for contingent constraints, and incorporates this new capability into a conflict-directed framework for efficient enumeration of solutions.

A *CSTNU with Decisions (CSTNUD)* extends a CSTNU by adding decisions (i.e., controllable choices) that are taken dynamically whatever any combination of truth-value assignments to the uncontrollable propositions and durations of contingent links (Zavatteri, 2017; Zavatteri & Viganò, 2018a). The dynamic controllability checking of CSTNUDs is sound and complete as it is done via TGAs.

When in a CSTNUD there is no temporal uncertainty and decisions can be taken in advance, then it is possible to check the dynamic controllability in a faster way (Cairo, Combi, Comin, Hunsberger, Posenato, Rizzi, & Zavatteri, 2017a).

Let us now informally discuss the expressivity of the CSTNUR formalism with respect to the CSTNUD one. Even though it seems plausible that resource allocations can be modeled as decisions, a polynomial-size encoding seems to be unfeasible.

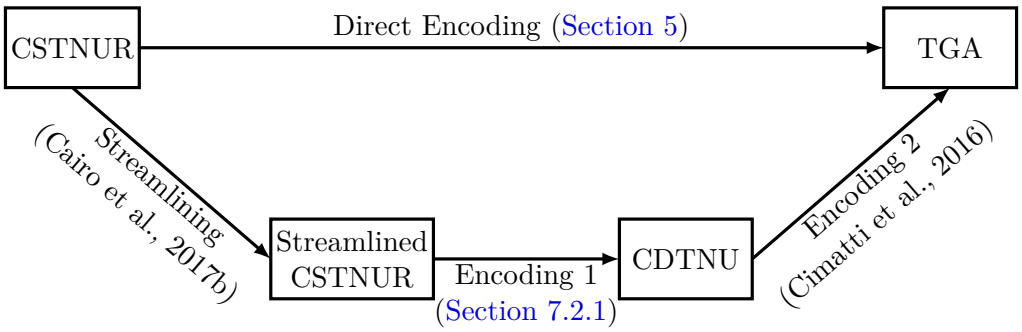


Figure 11: Deciding dynamic controllability of CSTNURs via TGAs: direct encoding (path above), reduction to CDTNUs (path below).

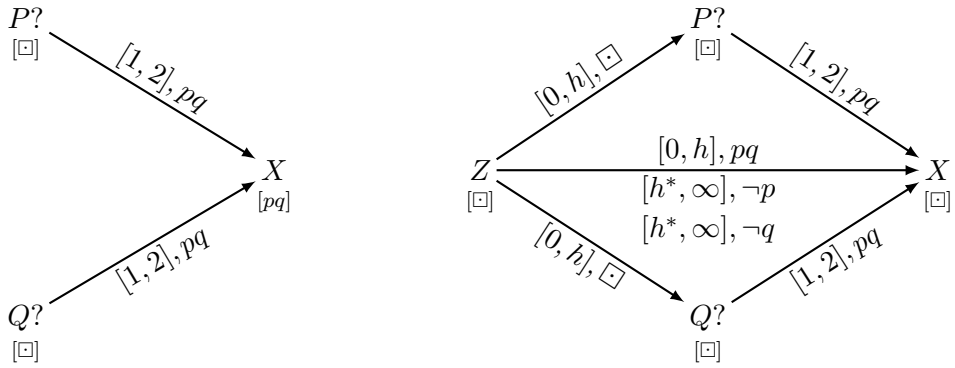
Indeed, if we consider a simple CSTNUR where resources have no TEs and there are no RRCs, it is possible to represent the instance as a CSTNUD one using decision variables to represent resource commitments to time points. In particular, it is necessary to represent any time point in the CSTNUD by replicating it as many times as the number of its possible resources and to represent the associated resource set as a set of decision nodes with a suitable set of constraints. Such constraints force the fact that if a decision is true (hence a resource is committed to the corresponding time point), all other decisions must be false and related time points must not be executed. Such set of constraints requires an exponential number of labeled values with respect to the number of resources. Therefore, the representation becomes quite cumbersome even in this simple case. If TEs and RRCs must be considered, it is open to state if it is possible to represent a CSTNUR instance as a CSTNUD one even with an exponential mapping size.

A *Conditional Disjunctive Temporal Network with Uncertainty (CDTNU)* extends a CSTNU by allowing disjunctive temporal constraints and disjunctive contingent links (Cimatti et al., 2016). CSTNURs are not more expressive than CDTNUs but they offer a compact language to model a temporal plan with resources and a direct encoding into TGAs. Indeed, if we employed CDTNUs we first should encode CSTNURs into CDTNUs and then encode CDTNUs into TGAs to decide dynamic controllability (double encoding). Figure 11 depicts the two different approaches for encoding a CSTNUR instance into a TGA, while Section 7.2.1 describes how to encode CSTNURs into CDTNUs.

7.2.1 Encoding CSTNURs into CDTNUs

Since resources can basically be seen as controllable (discrete) choices, one could wonder if a CSTNUR can be encoded into a *Conditional Disjunctive Temporal Networks with Uncertainty (CDTNU, (Cimatti et al., 2016))*, a formalism able to deal with uncontrollable choices, disjunctive uncontrollable durations and disjunctive constraints. The answer is yes and we prove it.

Before starting we summarize the *streamlined* model of CSTNs given in (Cairo et al., 2017b), which plays an important role in this encoding. Figure 12a gives an example of well-defined CSTN with three time points: $P?, Q?$ (always executed) and X (executed iff



(a) Example of CSTN. X is executed if and only if $p = q = \top$.

(b) Streamlined CSTN. X is always executed (within h if relevant, after h otherwise).

Figure 12: Streamlined models for conditional temporal networks: Removing labels from nodes preserving dynamic controllability. For this example, the deadline $h = 6$ and the delay $h^* = 7$ are good values to express "within" and "after" the horizon.

$p = q = \top$). Since labeling time points complicates proofs by adding further conditions such as "if the time point is executed" (i.e., if it is relevant), a streamlined model was proposed to convert a CSTN with labels on nodes into an equivalent streamlined CSTN with labels on constraints only. Figure 12b shows the streamlined representation of Figure 12a. The main idea is the following. We add a time point Z whose execution is fixed at 0. We compute an *horizon* value $h = M \times N$, where M is the maximum delay in the network and N the number of time points. For Figure 12a $M = 2$ and $N = 4$ (once we have added Z), therefore $h = 6$. We constrain every time point to occur within h (if relevant in the original CSTN), and after h otherwise. For Figure 12a "after h" is modeled by $h^* = h + 1 = 7$ (any $h^* > h$ is fine).

A streamlined CSTN is dynamically controllable if and only if the original CSTN is so (Cairo et al., 2017b). Therefore, "streamlining" a CSTN (but in general any temporal network subject to conditionals) preserves dynamic controllability (or uncontrollability) of the network getting another network in which all time points are always executed. Note that well-defined properties such as label honesty, constraint honesty and label coherence necessary for defining a CSTN instance become superfluous as they trivially hold in the streamlined CSTN (Cairo et al., 2017b). In what follows, we will consider streamlined models of the temporal networks under analysis obtained the same way of CSTNs (we just consider a bigger horizon to address upper bounds of contingent links).

Definition 24. (Cimatti et al., 2016) A *Conditional Disjunctive Temporal Network with Uncertainty (CDTNU)* is a tuple $\langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, \mathcal{P}, \mathcal{L} \rangle$, where

- $\mathcal{T}, L, \mathcal{OT}, \mathcal{O}, \mathcal{P}$ are the same as those given for a CSTNU (Definition 1).
- $\mathcal{C}$ is a set of constraints (ϕ, ℓ), where ϕ is an arbitrary Boolean combination of atoms of the form $Y - X \leq k$ for $X, Y \in \mathcal{T}$ and $k \in \mathbb{R}$ and $\ell \in \mathcal{P}^*$.

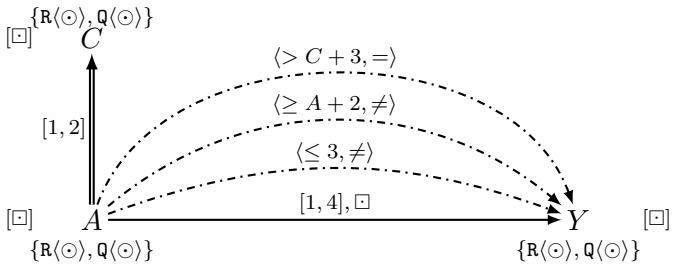


Figure 13: Fragment of CSTNUR

- $\mathcal{L}$ is a set of contingent links $(A, \mathcal{B}, C)$, where $A, C \in \mathcal{T}$ and $\mathcal{B}$ is a finite set of (disjoint) ranges $[x, y]$ such that $0 < x < y < \infty$.

When each contingent link specifies exactly one range and all constraints (ϕ, ℓ) are such that ϕ does not contain any disjunction, then the CDTNU boils down to a classic CSTNU.

For example, if $\mathcal{C}$ contains $\phi = ((Y - X \leq -3) \vee (Y - X \geq 4), p\neg q)$ it means that whenever p is true and q is false, then the controller must schedule Y and X in a way that must satisfy *either* $(Y - X \leq -3)$ *or* $(Y - X \geq 4)$. Instead, a contingent link $(A, \{[2, 3] \vee [5, 7]\}, C)$ means that once the controller executes A, the environment first chooses to assign *either* $[2, 3]$ *or* $[5, 7]$ to the contingent link and then it schedules C such that $C - A$ belongs to the chosen range.

We compact the notation for constraints and write

- $Y - X = k$ as a short for $Y - X \leq k$ and $X - Y \leq -k$, and
- $X = k$, $X \leq k$, and $X \geq k$ as shorts for $X - Z = k$, $X - Z \leq k$, $Z - X \leq -k$, where Z is the zero time point.

Consider the fragment of CSTNUR in Figure 13, where $h = 4$ and $h^* = 5$.

The first problem we come across is that of encoding resources and their commitment for time point executions. All *unlabeled* time points (observation ones included), contingent links and *labeled* constraints in the streamlined CSTNUR belong to the CDTNU too (note that contingent links do not turn disjunctive in the CDTNU). We assume to have an extra time point Z in the CSTNUR such that $\mathcal{R}(Z) = \{r^*\}$ and that no RRC will have Z neither as firing nor target time point. To model resources associated to time points, for each time point X (in the CSTNUR) such that $\mathcal{R}(X) = \{r_1, \ldots, r_n\}$, we add $|\mathcal{R}(X)|$ time points

$$X_{r_1}, \ldots, X_{r_n}$$

to the CDTNU such that $L(X_{r_1}) = \cdots = L(X_{r_n}) = \boxdot$. Each of these X_{r_i} can only be assigned two values: the same value that X gets during execution or h^*. If $X_{r_i} = X$, then it means that r_i is committed to execute X, whereas if $X_{r_i} = h^*$, then it means that r_i is not committed. Since we do not have a way to *exclude* some time point X_{r_i} from the execution, we follow the ideas of streamlined models and distinguish between executed or not executed time points by reasoning on the horizon. If a time point X_{r_i} is executed *within the horizon*,

then r_i was committed for X, whereas if X_{r_i} is executed *after the horizon*, then r_i was not committed.

Now, we must enforce that for each *non-contingent* time point X in the CSTNUR one and only one associated resource is committed for its execution. We model this condition in the CDTNU with the constraint

$$(\overbrace{(X_{r_1} = X \wedge \cdots \wedge X_{r_j} = h^*)}^{r_1 \text{ is committed for } X} \vee \cdots \vee \overbrace{(X_{r_1} = h^* \wedge \cdots \wedge X_{r_j} = X)}^{r_n \text{ is committed for } X}, \boxdot)$$

In Figure 13, we have that $\mathcal{R}(A) = \mathcal{R}(Y) = \{\mathtt{R}, \mathtt{Q}\}$, thus we add to the CDTNU the time points $Z, A_R, A_Q, C_R, C_Q, Y_R, Y_Q$ (recall that, $L(Z) = L(A_R) = L(A_Q) = L(C_Q) = L(C_Q) = L(Y_R) = L(Y_Q) = \boxdot$) and the constraints

- $(\underbrace{Z_{r^*} = Z \vee Z_{r^*} = h^*}_{\phi_1}, \boxdot)$ (but this constraint really doesn't matter)
- $(\underbrace{(\overbrace{(A_R = A \wedge A_Q = h^*)}^{\mathtt{R} \text{ is committed for } A} \vee \overbrace{(A_R = h^* \wedge A_Q = A)}^{\mathtt{Q} \text{ is committed for } A}}_{\phi_2}, \boxdot)$
- $(\underbrace{(\overbrace{(Y_R = Y \wedge Y_Q = h^*)}^{\mathtt{R} \text{ is committed for } Y} \vee \overbrace{(Y_R = h^* \wedge Y_Q = Y)}^{\mathtt{Q} \text{ is committed for } Y}}_{\phi_3}, \boxdot)$

to model the resource commitment for Z, A and Y.

For contingent time points we must commit the same resource that was committed for the related activation. We do so as follows. For each $(A, x, y, C) \in \mathcal{L}$ such that $\mathcal{R}(A) = \mathcal{R}(C) = \{r_1, \ldots, r_n\}$, we add the constraint

$$(\overbrace{(A = A_{r_1} \wedge C = C_{r_1}) \vee \cdots \vee (A = A_{r_n} \wedge C = C_{r_n})}^{\text{Commit the same resource for } C}, \boxdot)$$

In Figure 13, we have the contingent link $(A, 1, 2, C)$, therefore we add

$$(\underbrace{(A = A_R \wedge C = C_R) \vee (A = A_Q \wedge C = C_Q)}_{\phi_4}, \boxdot)$$

We are left to model RRCs in the CDTNU. We shorten the discussion focusing on the RRCs of Figure 13. Since an RRC between two time points X, Y either fires or doesn't fire depending on the order of execution of X and Y, we must hard-code a condition to understand which time point executes first. Note that this is necessary to handle limit cases where a temporal constraint $[0, 0]$ is specified between X and Y. Therefore, for every RRC $\langle X, \tau, \rho, Y\rangle$ we add a time point X_Y ($L(X_Y) = \boxdot$) and add the disjunctive constraint $(X_Y = X \vee X_Y = h^*, \boxdot)$. If $X_Y = X$, then X is executed before Y (even when $Y - X = 0$). If we have another RRC $\langle Y, \tau, \rho, X\rangle$, then we will add a Y_X and $(Y_X = X \vee Y_X = h^*, \boxdot)$ and also $((X_Y = X \wedge Y_X = h^*) \vee (X_Y = h^* \wedge Y_X = Y), \boxdot)$ (either X is before Y or the contrary). If X executes before Y, then the temporal constraint $X - Y \leq 0$ must hold. Therefore, for each $\langle X, \tau, \rho, Y\rangle$ we add the pair of constraints

- $(X_Y = X \Rightarrow X - Y \leq 0, \boxdot)$
- $(X_Y = h^* \Rightarrow X - Y \geq 0, \boxdot)$

Consider the RRC $A \to Y$ labeled by $\langle \leq 3, \neq \rangle$ in Figure 13. The condition we need to model is: *If A is executed before Y and the resources committed for A and Y are different, then Y must occur within global time 3*. That is, we add A_Y and the constraints

- $(\underbrace{A_Y = A \vee A_Y = h^*}_{\phi_5}, \boxdot)$
- $(\underbrace{A_Y = A \Rightarrow A - Y \leq 0}_{\phi_6}, \boxdot)$
- $(\underbrace{A_Y = h^* \Rightarrow A - Y \geq 0}_{\phi_7}, \boxdot)$
- $(\underbrace{A_Y = A \wedge \overbrace{((A_R = A \wedge Y_Q = Y) \vee (A_Q = A \wedge Y_R = Y))}^{\text{Resources committed for } A \text{ and } Y \text{ are different}} \Rightarrow Y \leq 3}_{\phi_8}, \boxdot)$

Consider RRC $A \to Y$ labeled by $\langle \geq A + 2, = \rangle$ in Figure 13. The condition we need to model is: *If A is executed before Y and the resources committed for A and Y are equal, then Y must be executed after minimum 2 since A.* That is, we add the constraint

$$(\underbrace{A_Y = A \wedge \overbrace{((A_R = A \wedge Y_R = Y) \vee (A_Q = A \wedge Y_Q = Y))}^{\text{Resources committed for } A \text{ and } Y \text{ are equal}} \Rightarrow Y \geq A + 2}_{\phi_9}, \boxdot)$$

Finally, consider the RRC $A \to Y$ labeled by $\langle > C + 3, = \rangle$ in Figure 13. The condition we need to model is: *If A is executed before Y and the resources committed for A and Y are equal, then (if C has already been executed, then Y must be executed after 3 since C), whereas (if C has not been executed, then no solution exists).* Since once fired, RRCs involve the execution of Y, we encode "C has already been executed" as "C is executed *before* Y", and "C has not been executed yet" as "C is executed *after* Y" ("whereas" here means "and"). Moreover, the "no solution exists" is because if C is still unexecuted by the time Y executes, then equal resources have associated temporal expressions that are not satisfied by the current time. Therefore, we add C_Y and the constraints

- $(\underbrace{C_Y = C \vee C_Y = h^*}_{\phi_{10}}, \boxdot)$
- $(\underbrace{C_Y = C \Rightarrow C - Y \leq 0}_{\phi_{11}}, \boxdot)$
- $(\underbrace{C_Y = h^* \Rightarrow C - Y \geq 0}_{\phi_{12}}, \boxdot)$

- (ϕ_{13}, ℓ) is $((A_Y = A \wedge \overbrace{((A_R = A \wedge Y_R = Y) \vee (A_Q = A \wedge Y_Q = Y))}^{\text{Resources committed for } A \text{ and } C \text{ are equal}}) \Rightarrow$
  $(\underbrace{(\underbrace{(C_Y = C \Rightarrow Y - C \leq 3)}_{\text{First case}} \wedge \underbrace{(C_Y = h^* \Rightarrow \underbrace{Y - Y \leq -1}_{\text{No sol. exists}})}_{\text{Second case}}), \boxdot)$

Note that "no solution exists" is modeled as a negative self loop. In this example, we used -1 as weight for $Y \to Y$, but any negative real value or any unsatisfiable constraint fulfills this purpose: "break the execution by using this".

Similar encodings apply for the other cases of RRCs with respect to the specific θ.

Finally, $\mathcal{C}$ consists of the native temporal constraints of the initial CSTNUR plus $(\phi_1, \boxdot) \wedge \cdots \wedge (\phi_{13}, \boxdot)$ that can be compacted as $(\texttt{CNF}((\phi_1) \wedge \cdots \wedge (\phi_{13})), \boxdot)$.

After that, dynamic controllability of the CDTNU can be checked by using the methods in (Cimatti et al., 2016). We point out that since resource commitments are Boolean conditions (any resource for any time point is either committed or not), when computing the conjunctive normal forms, we can safely impose that $\neg(X_{r_i} = X)$ is equivalent to $X_{r_i} = h^*$ (like we did for clocks modeling Boolean propositions in the TGA encodings where $\neg(\texttt{bP} < \hat{\texttt{c}})$ becomes $\texttt{bP} = \hat{\texttt{c}}$).

Such an encoding keeps a polynomial number of time points and constraints with respect to the size of the initial CSTNUR.

7.3 Managing Access Control Policies under Uncertainty

Shah, Conrad, and Williams (2009) proposed *Chaski*, an executive that dynamically dispatches plans with task assignment over heterogeneous, cooperative agents, represented by a *Temporal Constraint Satisfaction Network (TCSN)*, an extension of STNs where each edge can be labeled by a disjunction of non-overlapping ranges (Dechter et al., 1991). The focus of authors is on providing a compact representation of constraints, thus improving the performance of scheduling. In the paper, the authors considered only one rule for assigning agents to tasks, *agent occupancy constraints*, meaning that each agent may only perform one activity at a time. Even if their algorithms seems to be able to manage different assignment rules, there is no a formal definition of such rules and it is not specified whether they can depend on temporal aspects or they can be modified at runtime like our RRCs.

Combi, Viganò, and Zavatteri (2016) proposed a model where a workflow and a fragment of Temporal Role-Based Access Control (TRBAC) (Bertino, Bonatti, & Ferrari, 2001) are encoded into an STNU, and security policies are modeled by security constraints (SCs) along with security constraint propagation rules (SCPRs) that propagate them depending on which user is executing which time point. Dynamic controllability checking for this augmented network is addressed in this paper where the previously proposed SCs and SCPRs have been evolved into TEs and RRCs, respectively.

Access Controlled Temporal Networks (ACTNs) (Combi, Posenato, Viganò, & Zavatteri, 2017) face the same issue we studied in this paper but do not employ TEs and RRCs. Indeed, ACTNs specify a static set of resource constraints. Dynamic controllability checking is addressed via TGAs by using a different encoding, where the resource constraints are verified in the winning path. Depending on the arising scenario, the relevant constraints must always be satisfied. ACTNs are able to handle contingent durations, conditional constraints, and

disjunctive resource constraints with respect to the authorization policy defining the security part. This work is completely different as RRCs are a kind of "directional constraints" which might, or might not, be taken into consideration depending on the order in which time points are executed.

Vasilikos, Nielson, and Nielson (2017) proposed a network of Timed Automata to model distributed systems and provide Behavior Timed Computational Tree Logics (BTCTL), an extension of Timed Computation Tree Logic (TCTL) (Alur, Courcoubetis, & Dill, 1993) to express time-dependent access control policies. Such a logic allows the expressions of security policies in which temporal, data and information flow aspects must be considered together. In particular, the authors proposed a reduction of a fragment of Behavior TCTL (BTCTL) into TCTL$^+$ (a variation of TCTL) that can be validated using Uppaal. In this model no uncontrollable part is supported. Our work focuses on resource allocation and deals with uncontrollable parts, an issue which is managed by TGAs instead of classic TAs. Furthermore, besides for the fact that CSTNURs do not deal with any data, the main difference is that the proposal in (Vasilikos et al., 2017) enforces security policies at system level, whereas we synthesize a controller that avoids breaching security policies when the constraints of the system would allow some execution to do so. We also showed that CSTNURs can be encoded into (native) TGAs without using extra variables and that TCTL model checking is enough for our purpose.

Zavatteri, Combi, Posenato, and Viganò (2017) proposed an initial approach to check weak, strong and dynamic controllability for access controlled workflows under conditional uncertainty by mapping workflow paths to *Constraint Networks (CNs)* (Dechter, 2003) and reasoning on the intersection of common parts. The proposed approach pointed out that dynamic controllability might be a matter of how the components of the workflow are ordered, an hypothesis that was later confirmed with the proposal of *Constraint Networks Under Conditional Uncertainty (CNCUs)* (Zavatteri & Viganò, 2018b; Zavatteri & Viganò, 2019). However, both these last two works do not deal with temporal constraints.

8. Conclusions

In this paper, we proposed Conditional Simple Temporal Networks with Uncertainty and Resources to manage all together temporal constraints with uncertainty, uncertain conditions and resource assignments. Resource assignments can be subject to further temporal constraints that can contain an explicit reference to the execution time. Indeed, resources may be associated to Temporal Expressions (TE) to specify when resources are available to be assigned to time points. Moreover, it is possible to specify also Runtime Resource Constraints (RRCs), a new class of temporal constraints, for refining TEs associated to resources in real time depending on the specific execution. We described and discussed the CSTNUR model through a real-world motivating example representing a round-trip flight process in which we enforced the *flight time limitations and rest requirements for pilots* according to the official FAA regulations.

A CSTNUR instance is dynamically controllable if there exists an execution strategy (specified as an RTED strategy) to execute all time points by assigning an executing agent to each of them and satisfying all temporal constraints, all applicable TEs and RRCs, no matter

which truth values for propositions and durations for contingent links are incrementally revealed over time.

To check the dynamic controllability property of CSTNUR instances, we proposed a new mapping from CSTNURs into Time Game Automata by extending the encoding proposed in (Cimatti et al., 2016) for CSTNUs. We proved that such mapping is correct and can be determined in polynomial time.

Appendix A. A possible implementation with Uppaal-TIGA

Uppaal is an integrated software tool for the modeling, validation and verification of real-time systems modeled as networks of Timed Automata, extended with data types (bounded integers, Boolean variables, arrays, etc.). Uppaal-TIGA is an extension of Uppaal implementing the first efficient on-the-fly algorithm for solving games based on TGAs with respect to reachability and safety properties (Behrmann et al., 2007).

As a proof-of-concept, we wrote the specification of the TGA encoding the CSTNUR depicted in Figure 6 and ran Uppaal-TIGA to answer to the decision problem of dynamic controllability (Zavatteri, 2019).We took advantage of Boolean variables to represent propositions and the RA relation[6].

We used a FreeBSD virtual machine running on top of a VMWare ESXi using a physical machine equipped with an Intel i7 2.80GHz and 20GB of RAM for the experimental evaluation. The VM was assigned 16GB of RAM[7] and full CPU power.

We verified that the CSTNUR in Figure 6 is dynamically controllable. The model checking phase took 207 minutes and 28 seconds to synthesize a 1.6MB memoryless execution strategy as a *certificate of YES* for this decision problem. Such a strategy consists of statements like *state* $\rightarrow$ *action*, where *state* abstracts conditions over locations, clock constraints (and Boolean variables), whereas *action* says either to take a specific transition or to wait. Figure 14 shows how the TGA encoding the CSTNUR in Figure 6 looks like in Uppaal-TIGA.

6. For each proposition p, $p = \top$ (resp., $p = \bot$) means $\texttt{bP} = \hat{\texttt{c}}$ (resp., $\texttt{bP} < \hat{\texttt{c}}$). For each $(u, X) \in RA$, $uX = \top$ means $\texttt{uX} > \hat{\texttt{c}}$ if $\texttt{cX} < \hat{\texttt{c}}$ (u executed X), $\texttt{uX} = \hat{\texttt{c}}$ if $\texttt{cX} = \hat{\texttt{c}}$ (u is available), whereas $uX = \bot$ means $\texttt{uX} = \texttt{cX}$ if $\texttt{cX} < \hat{\texttt{c}}$ (u did not execute X) or $\texttt{uX} < \texttt{cX}$ if $\texttt{cX} = \hat{\texttt{c}}$ (u has been blocked).
7. Plenty, since Uppaal-TIGA is compiled for 32bit architectures.

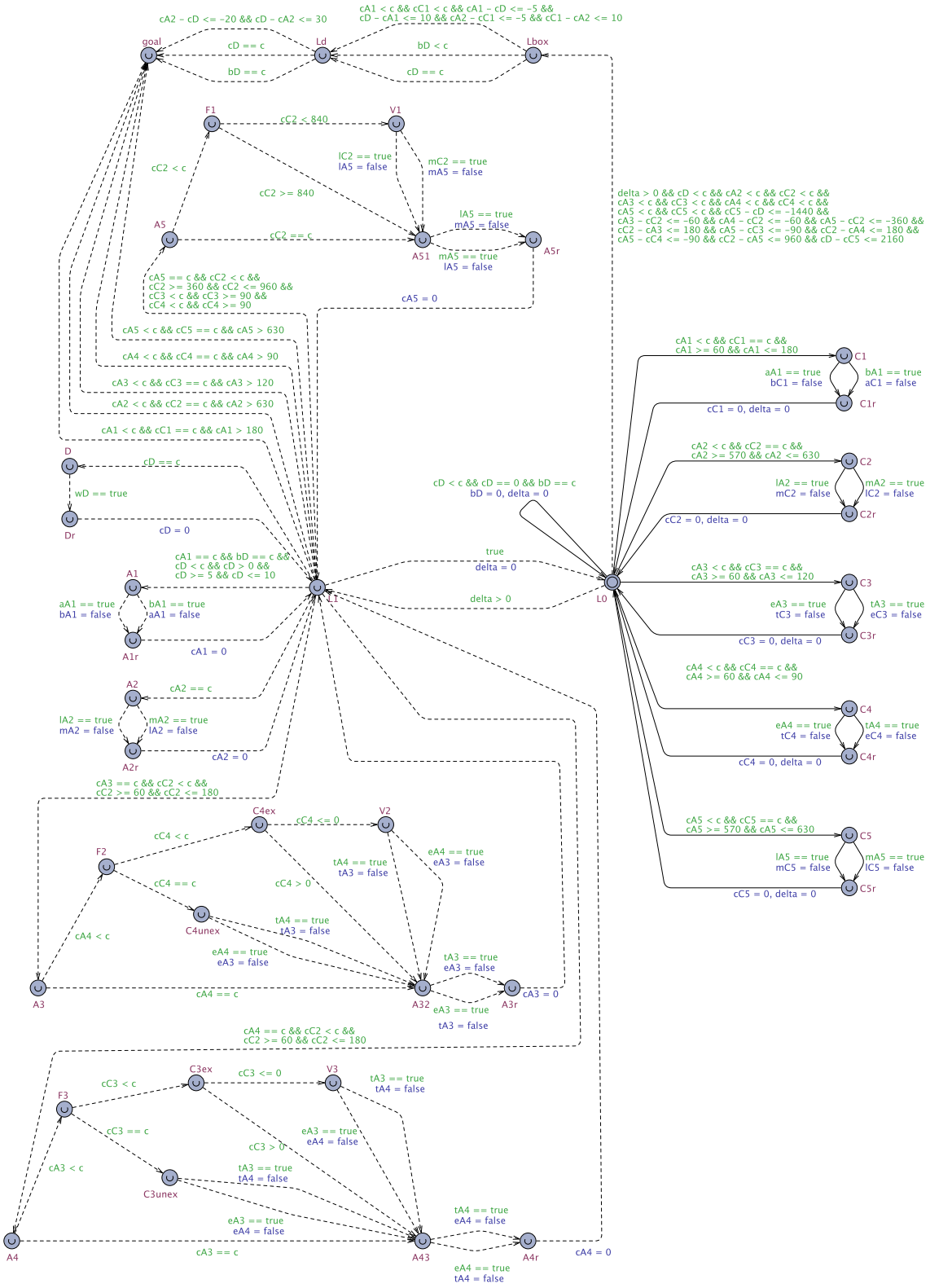


Figure 14: Modeling and validation of Figure 6 with Uppaal-TIGA.

References

Alur, R., Courcoubetis, C., & Dill, D. (1993). Model-checking in dense real-time. *Inf. Comput.*, *104*(1), 2–34, doi: 10.1006/inco.1993.1024.

Alur, R., & Dill, D. L. (1994). A theory of timed automata. *Theor. Comput. Sci.*, *126*(2), 183–235, doi: 10.1016/0304-3975(94)90010-8.

Avanes, A., & Freytag, J.-C. (2008). Adaptive workflow scheduling under resource allocation constraints and network dynamics. *Proc. VLDB Endow.*, *1*(2), 1631–1637, doi: 10.14778/1454159.1454238.

Balas, E. (1979). Disjunctive programming. In Hammer, P., Johnson, E., & Korte, B. (Eds.), *Discrete Optimization II*, Vol. 5 of *Annals of Discrete Mathematics*, pp. 3 – 51. Elsevier, doi: 10.1016/S0167-5060(08)70342-X.

Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K. G., & Lime, D. (2007). Uppaal-tiga: Time for playing games!. In *Computer Aided Verification, 19th International Conference, CAV 2007*, pp. 121–125, doi: 10.1007/978-3-540-73368-3_14.

Bertino, E., Bonatti, P. A., & Ferrari, E. (2001). TRBAC: A temporal role-based access control model. *ACM Trans. Inf. Syst. Secur.*, *4*(3), 191–233, doi: 10.1145/501978.501979.

Bettini, C., Wang, X. S., & Jajodia, S. (2002). Temporal reasoning in workflow systems. *Distributed and Parallel Databases*, *11*(3), 269–306, doi: 10.1023/A:1014048800604.

Cairo, M., Combi, C., Comin, C., Hunsberger, L., Posenato, R., Rizzi, R., & Zavatteri, M. (2017a). Incorporating decision nodes into conditional simple temporal networks. In Schewe, S., Schneider, T., & Wijsen, J. (Eds.), *24th International Symposium on Temporal Representation and Reasoning, TIME 2017*, Vol. 90 of *LIPIcs*, pp. 9:1–9:17, doi: 10.4230/LIPIcs.TIME.2017.9.

Cairo, M., Hunsberger, L., Posenato, R., & Rizzi, R. (2017b). A streamlined model of conditional simple temporal networks. In Schewe, S., Schneider, T., & Wijsen, J. (Eds.), *24th International Symposium on Temporal Representation and Reasoning, TIME 2017*, Vol. 90 of *LIPIcs*, pp. 10:1–10:19, doi: 10.4230/LIPIcs.TIME.2017.10.

Cairo, M., & Rizzi, R. (2016). Dynamic controllability of conditional simple temporal networks is pspace-complete. In *23nd International Symposium on Temporal Representation and Reasoning, TIME 2016*.

Chinn, S. J., & Madey, G. R. (2000). Temporal representation and reasoning for workflow in engineering design change review. *IEEE Transactions on Engineering Management*, *47*(4), 485–492.

Cimatti, A., Do, M., Micheli, A., Roveri, M., & Smith, D. E. (2018). Strong temporal planning with uncontrollable durations. *Artificial Intelligence*, *256*, 1–34, doi: 10.1016/j.artint.2017.11.006.

Cimatti, A., Hunsberger, L., Micheli, A., Posenato, R., & Roveri, M. (2014). Sound and complete algorithms for checking the dynamic controllability of temporal networks with uncertainty, disjunction and observation. In *21st International Symposium on*

Temporal Representation and Reasoning, TIME 2014, pp. 27–36, doi: 10.1109/TIME.2014.21.

Cimatti, A., Hunsberger, L., Micheli, A., Posenato, R., & Roveri, M. (2016). Dynamic controllability via Timed Game Automata. *Acta Informatica*, *53*(6–8), 681–722, doi: 10.1007/s00236-016-0257-2.

Cimatti, A., Hunsberger, L., Micheli, A., & Roveri, M. (2014). Using timed game automata to synthesize execution strategies for simple temporal networks with uncertainty. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pp. 2242–2249.

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

Combi, C., Hunsberger, L., & Posenato, R. (2014b). An algorithm for checking the dynamic controllability of a conditional simple temporal network with uncertainty - revisited. In *Proceedings of 5th International Conference on Agents and Artificial Intelligence, ICAART 2013, Revised Selected Papers*, pp. 314–331, doi: 10.1007/978-3-662-44440-5_19.

Combi, C., & Posenato, R. (2009). Controllability in temporal conceptual workflow schemata. In *Proceedings of 7th International Conference on Business Process Management, BPM 2009*, Vol. 5701 of *LNCS*, pp. 64–79. Springer, doi: 10.1007/978-3-642-03848-8_6.

Combi, C., & Posenato, R. (2010). Towards temporal controllabilities for workflow schemata. In *Proceedings of 17th International Symposium on Temporal Representation and Reasoning, TIME 2010*, pp. 129–136, doi: 10.1109/TIME.2010.17.

Combi, C., Posenato, R., Viganò, L., & Zavatteri, M. (2017). Access controlled temporal networks. In van den Herik, H. J., Rocha, A. P., & Filipe, J. (Eds.), *Proceedings of the 9th International Conference on Agents and Artificial Intelligence, ICAART 2017*, pp. 118–131. SciTePress, doi: 10.5220/0006185701180131.

Combi, C., Viganò, L., & Zavatteri, M. (2016). Security constraints in temporal role-based access-controlled workflows. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, CODASPY '16. ACM.

Conrad, P. R., & Williams, B. C. (2011). Drake: An efficient executive for temporal plans with choice. *J. Artif. Int. Res.*, *42*(1), 607–659.

Dechter, R. (2003). *Constraint processing*. Elsevier.

Dechter, R., Meiri, I., & Pearl, J. (1991). Temporal constraint networks. *Artif. Intell.*, *49*(1-3), 61–95, doi: 10.1016/0004-3702(91)90006-6.

Eder, J., Gruber, W., & Panagos, E. (2000). Temporal Modeling of Workflows with Conditional Execution Paths BT - Database and Expert Systems Applications. In *Database and Expert Systems Applications*, Vol. LNCS 1873, pp. 243–253. Springer, Berlin, Heidelberg.

FAR (2019). *Electronic Code of Federal Regulations*. https://www.ecfr.gov/cgi-bin/text-idx?view=text&node=14:2.0.1.3.10#se14.2.91_11059.

Fox, M., & Long, D. (2003). PDDL2.1: an extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res.*, *20*, 61–124, doi: 10.1613/jair.1129.

Hopcroft, J. E., & Ullman, J. D. (1979). *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley.

Hunsberger, L. (2009). Fixing the semantics for dynamic controllability and providing a more practical characterization of dynamic execution strategies. In *Proceedings of 16th International Symposium on Temporal Representation and Reasoning, TIME 2009*, pp. 155–162, doi: 10.1109/TIME.2009.25.

Hunsberger, L., & Posenato, R. (2018a). Dynamic Controllability Checking for Conditional Simple Temporal Networks with Uncertainty: New Sound-and-Complete Algorithms based on Constraint Propagation. In Alechina, N., Nørvåg, K., & Penczek, W. (Eds.), *25th International Symposium on Temporal Representation and Reasoning, TIME 2018*, Vol. 120 of *LIPIcs*, pp. 14:1–14:17, doi: 10.4230/LIPIcs.TIME.2018.14.

Hunsberger, L., & Posenato, R. (2018b). Sound-and-Complete Algorithms for Checking the Dynamic Controllability of Conditional Simple Temporal Networks with Uncertainty. In Alechina, N., Nørvåg, K., & Penczek, W. (Eds.), *25th International Symposium on Temporal Representation and Reasoning (TIME 2018)*, Vol. 120 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 14:1–14:17, doi: 10.4230/LIPIcs.TIME.2018.14.

Hunsberger, L., Posenato, R., & Combi, C. (2012). The Dynamic Controllability of Conditional STNs with Uncertainty. In *PlanEx at ICAPS 2012*, pp. 1–8.

Hunsberger, L., Posenato, R., & Combi, C. (2015). A sound-and-complete propagation-based algorithm for checking the dynamic consistency of conditional simple temporal networks. In Grandi, F., Lange, M., & Lomuscio, A. (Eds.), *22nd International Symposium on Temporal Representation and Reasoning, TIME 2015*, pp. 4–18. IEEE CPS, doi: 10.1109/TIME.2015.26.

Kafeza, E., & Karlapalem, K. (2000). Gaining control over time in workflow management applications. In *Proceedings of 11th International Conference on Database and Expert Systems Applications, DEXA 2000*, pp. 232–242, doi: 10.1007/3-540-44469-6_22.

Kim, P., Williams, B. C., & Abramson, M. (2001). Executing reactive, model-based programs through graph-based temporal planning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence, IJCAI 2001*, Vol. 1, pp. 487–493.

Léauté, T., & Williams, B. C. (2005). Coordinating agile systems through the model-based execution of temporal plans. In *Proceedings of the 20th national conference on Artificial intelligence, AAAI'05*, pp. 114–120.

Levine, S. J., & Williams, B. C. (2014). Concurrent plan recognition and execution for human-robot teams. In Chien, S. A., Do, M. B., Fern, A., & Ruml, W. (Eds.), *Proceedings of the 24th International Conference on Automated Planning and Scheduling, ICAPS 2014*.

Maler, O., Pnueli, A., & Sifakis, J. (1995). *On the synthesis of discrete controllers for timed systems*, pp. 229–242. Springer, doi: 10.1007/3-540-59042-0_76.

Morris, P. H., & Muscettola, N. (2005). Temporal dynamic controllability revisited. In *Proceedings 20th National Conference on Artificial Intelligence and the 27th Innovative Applications of Artificial Intelligence Conference*, pp. 1193–1198.

Morris, P. H., Muscettola, N., & Vidal, T. (2001). Dynamic control of plans with temporal uncertainty. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001*, pp. 494–502.

Posenato, R., Zerbato, F., & Combi, C. (2018). Managing Decision Tasks and Events in Time-Aware Business Process Models. In *Business Process Management. BPM 2018*, Vol. 11080 of *LNCS*, pp. 102–118. Springer, doi: 10.1007/978-3-319-98648-7_7.

Qi, C., Wang, D., Muñoz-Avila, H., Zhao, P., & Wang, H. (2017). Hierarchical task network planning with resources and temporal constraints. *Knowledge-Based Systems*, *133*, 17–32, doi: 10.1016/j.knosys.2017.06.036.

Shah, J. A., Conrad, P. R., & Williams, B. C. (2009). Fast distributed multi-agent plan execution with dynamic task assignment and scheduling. In Gerevini, A., Howe, A. E., Cesta, A., & Refanidis, I. (Eds.), *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009*. AAAI.

Stergiou, K., & Koubarakis, M. (2000). Backtracking algorithms for disjunctions of temporal constraints. *Artif. Intell.*, *120*(1), 81–117, doi: 10.1016/S0004-3702(00)00019-9.

Tsamardinos, I., Vidal, T., & Pollack, M. E. (2003). CTP: A new constraint-based formalism for conditional, temporal planning. *Constraints*, *8*(4), 365–388, doi: 10.1023/A:1025894003623.

Vasilikos, P., Nielson, F., & Nielson, H. R. (2017). Time Dependent Policy-Based Access Control. In Schewe, S., Schneider, T., & Wijsen, J. (Eds.), *24th International Symposium on Temporal Representation and Reasoning, TIME 2017*, Vol. 90 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 21:1–21:18, doi: 10.4230/LIPIcs.TIME.2017.21.

Vidal, T., & Fargier, H. (1997). Contingent Durations in Temporal CSPs: From Consistency to Controllabilities. In *Proc. 4th Int. Work. Temporal Represent. Reason. (TIME '97)*.

Vidal, T., & Fargier, H. (1999). Handling contingency in temporal constraint networks: from consistency to controllabilities. *J. Exp. Theor. Artif. Intell.*, *11*(1), 23–45, doi: 10.1080/095281399146607.

Vidal, T., & Ghallab, M. (1996). Dealing with uncertain durations in temporal constraint networks dedicated to planning. In *European Conference on Artificial Intelligence (ECAI)*, pp. 48–54. PITMAN.

Watahiki, K., Ishikawa, F., & Hiraishi, K. (2011). Formal verification of business processes with temporal and resource constraints. In *2011 IEEE International Conference on Systems, Man, and Cybernetics*, pp. 1173–1180, doi: 10.1109/ICSMC.2011.6083857.

Yu, P., Fang, C., & Williams, B. C. (2014). Resolving uncontrollable conditional temporal problems using continuous relaxations. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling, ICAPS 2014*.

Yu, P., & Williams, B. C. (2013). Continuously relaxing over-constrained conditional temporal problems through generalized conflict learning and resolution. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence, IJCAI 2013*, pp. 2429–2436.

Zavatteri, M. (2017). Conditional simple temporal networks with uncertainty and decisions. In *24th International Symposium on Temporal Representation and Reasoning, TIME 2017*, Vol. 90 of *LIPIcs*, pp. 23:1–23:17, doi: 10.4230/LIPIcs.TIME.2017.23.

Zavatteri, M. (2019). *Example of a CSTNUR encoded as TGA*. http://regis.di.univr.it/FlightExample.tar.bz2.

Zavatteri, M., Combi, C., Posenato, R., & Viganò, L. (2017). Weak, strong and dynamic controllability of access-controlled workflows under conditional uncertainty. In *Business Process Management (BPM 2017)*, pp. 235–251. Springer, doi: 10.1007/978-3-319-65000-5_14.

Zavatteri, M., & Viganò, L. (2018a). Conditional simple temporal networks with uncertainty and decisions. *Theor. Comput. Sci.*, (In press), , doi: 10.1016/j.tcs.2018.09.023.

Zavatteri, M., & Viganò, L. (2018b). Constraint networks under conditional uncertainty. In *Proceedings of the 10th International Conference on Agents and Artificial Intelligence, ICAART 2018*, pp. 41–52. INSTICC, SciTePress, doi: 10.5220/0006553400410052.

Zavatteri, M., & Viganò, L. (2019). Conditional uncertainty in constraint networks. In *Agents and Artificial Intelligence (ICAART 2018)*, pp. 130–160. Springer, doi: 10.1007/978-3-030-05453-3_7.