Conditional Simple Temporal Networks with Uncertainty and Resources

Carlo Combi
Dipartimento di Informatica, Università di Verona, Italy
carlo.combi@univr.it

Roberto Posenato
Dipartimento di Informatica, Università di Verona, Italy
roberto.posenato@univr.it

Luca Viganò
Department of Informatics, King’s College London, UK
luca.vigano@kcl.ac.uk

Matteo Zavatteri
Dipartimento di Informatica, Università di Verona, Italy
matteo.zavatteri@univr.it

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 [33, 25, 11, 7, 16, 18, 19, 12]. 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 \[3, 49, 41\]. 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.

Contributions

The novelty of our proposal is three-fold:

1. CSTNURs allow us to represent:

   (i) 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);
   (ii) execution scenarios that are not under the control of the system (i.e., uncontrollable conditions);
   (iii) 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 [28, 15, 13, 14], 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 in [13] to consider resources and RRCs. We formally prove the correctness of the proposed encoding and discuss its complexity.

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 \[40\]. Such a notation has been completed by adding available resources for tasks (\{Alice, 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 Deicing? = ⊤ (i.e., Deicing? is true) then the Deicing process starts after minimum 5 minutes and within 10 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? = ⊥ (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 FAA regulations for flight time limitations and rest requirements\[2\] 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 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.

\[1\] 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.

\[2\] https://www.ecfr.gov/cgi-bin/text-idx?view=text&node=14:2.0.1.3.10#se14.2.91_11059
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) [24] 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) [24].

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 [18, 46, 47, 39]. 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 [38].

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 $P$ of propositional letters, a label $\ell$ is any conjunction of literals, where a literal is either a propositional letter $p \in P$ or its negation $\neg p$. The empty label is denoted by $\Box$. The label universe of $P$, denoted by $P^*$, is the set of all labels whose literals are drawn from $P$; e.g., the label universe of $P = \{p, q\}$ is $P^* = \{\Box, p, q, \neg p, \neg q, p \land q, p \land \neg q, \neg p \land q, \neg p \land \neg q\}$. Two labels $\ell_1, \ell_2 \in P^*$ are consistent if and only if their conjunction $\ell_1 \land \ell_2$ is satisfiable and a label $\ell_1$ entails a label $\ell_2$ (written $\ell_1 \Rightarrow \ell_2$) if and only if all literals in $\ell_2$ appear in $\ell_1$ too (i.e., if $\ell_1$ is more specific than $\ell_2$). For instance, if $\ell_1 = p \land \neg q$ and $\ell_2 = p$, then $\ell_1$ and $\ell_2$ are consistent since $p \land \neg q \land p$ is satisfiable, and $\ell_1$ entails $\ell_2$ since $p \land \neg q \Rightarrow p$. The vice versa does not hold in general (e.g., $p \not\Rightarrow p \land \neg q$).

The following definition summarizes the definition of CSTNUs [31, 17] and some basic properties presented in [32] that must hold in such networks.
**Definition 1** (CSTNU). A *Conditional Simple Temporal Network with Uncertainty (CSTNU)* is a tuple \( \langle T, P, L, \mathcal{OT}, O, C, \mathcal{L} \rangle \), where:

1. \( T = \{X, Y, \ldots \} \) is a finite set of *time points* (i.e., variables with continuous domain).
2. \( P = \{p, q, \ldots \} \) is a finite set of propositional letters.
3. \( L: T \rightarrow P^* \) is a function assigning a label to each time point \( X \in T \).
4. \( \mathcal{OT} \subseteq T \) is a set of *observation time points*.
5. \( O: P \rightarrow \mathcal{OT} \) is a bijection associating a unique observation time point to each propositional letter.
6. \( C \) is a set of *labeled constraints* each one having the form \( (l \leq Y - X \leq u, \ell) \), where \( X, Y \in T \), \( l, u \in \mathbb{R} \) with \( l \leq u \) and \( \ell \in P^* \). If, given \( X, Y \in T \) and \( \ell \in P^* \), there exist two constraints \( (l_1 \leq Y - X \leq u_1, \ell) \) and \( (l_2 \leq Y - X \leq u_2, \ell) \) in \( C \), 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 \( \ell \).
7. \( \mathcal{L} \) is a set of *contingent links* each having the form \( (A, x, y, C) \), where \( A, C \in T \) are different time points (written \( A \neq C \)), \( 0 \leq x < Y < \infty \) 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 \( A_1 \neq A_2 \) we have that \( C_1 \neq C_2 \).
8. For each constraint \( (l \leq Y - X \leq u, \ell) \in C \), we have that \( \ell \Rightarrow L(Y) \land L(X) \). This property is called *constraint label coherence* \([32]\).
9. For each literal \( p \) or \( \neg p \) appearing in \( \ell \), we have that \( \ell \Rightarrow L(O(p)) \). This property is called *constraint label honesty* \([32]\).
10. For each \( X \in T \), if literal \( p \) or \( \neg p \) appears in \( L(X) \), then \( O(p) \) has to occur before \( X \), i.e., \( (\epsilon \leq X - O(p) \leq +\infty, L(X)) \in C \) for some \( \epsilon > 0 \), where \( \epsilon \) is a real constant modeling a non-instantaneous reaction time. This property is called *time point label honesty* \([32]\).

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.  

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*. 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* $\text{ctrl}$ and the *environment* $\text{env}$ [14]. 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 $\text{ctrl}$ and $\text{env}$, and instantaneous reactions for $\text{env}$ only. $\text{ctrl}$’s RTEDs model a strategy for executing all relevant non-contingent time points of the network, whereas $\text{env}$’s RTEDs model an unknown strategy for executing contingent time points. Likewise, the instantaneous reactions represent an unknown strategy for both reacting to $\text{ctrl}$’s actions and setting the truth values of the propositions.

The purpose of $\text{ctrl}$ 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

---

4 This representation is a simplified one obtained considering the mapping presented in [40] where nodes that have [0, 0] distance have been collapsed in a single CSTNU node.
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 $\mathcal{PS} = (\text{Executed, Assigned})$, 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 $\text{Executed}_T = \{X \mid (X, k) \in \text{Executed}\}$ and $\text{Executed}_R = \{k \mid (X, k) \in \text{Executed}\}$, respectively. If $(X, k) \in \text{Executed}$, then $\text{time}(X) = k$. $\text{last}(\text{Executed}) = \max \{v \mid v \in \text{Executed}_R\}$ represents the time instant of the last executed time point; when $\text{Executed} = \emptyset$, we assume $\text{last}(\text{Executed}) = -\infty$.

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

A partial schedule is called locally consistent (formerly, respectful in [13]) if all constraints $(x \leq Y - X \leq y, \ell)$ of the considered CSTNU such that $\ell_{\text{cps}} \Rightarrow \ell$ and $Y, X \in \text{Executed}_T$ 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, $\Delta_{\text{ctrl}}$, specifies which action has to be performed by ctrl during an execution (represented by a considered $\mathcal{PS}$). It has two forms: wait or $(t, \text{NonContingent})$.

- $\Delta_{\text{ctrl}} = \text{wait}$ 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).

- $\Delta_{\text{ctrl}} = (t, \text{NonContingent})$ 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 $t > \text{last}(\text{Executed})$. NonContingent is a non-empty set of unexecuted non-contingent time points (i.e., $\text{NonContingent} \subseteq T \setminus \text{Executed}_T$ and $\text{NonContingent} \neq \emptyset$) and $\ell_{\text{cps}} \Rightarrow L(X)$ for each $X \in \text{NonContingent}$ (i.e., $X$ must be considered in the current partial scenario).

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

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

- $\Delta_{\text{env}} = \text{wait}$ 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.
The set of all RTEDs for \( \text{env} \) is denoted by \( \Delta_{\text{env}} \).

We now consider the instantaneous reactions for \( \text{env} \). Such reactions entail that \( \text{env} \) operates instantaneously on some uncontrollable part(s). For example, \( \text{env} \) 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 \( \text{env} \)). Let \( \mathcal{PS} = (\text{Executed, Assigned}) \) be a partial schedule. Let \( \chi^0 \) be the set of contingent time points \( C_i \) such that \( A_i \in \text{Executed}_\tau \) and \( \text{time}(A_i) + y_i = \text{last}(\text{Executed}) \), i.e., their execution windows terminate precisely at \( \text{last}(\text{Executed}) \). Let \( \chi^* \) be any (possibly empty) subset of the contingent time points \( C_i \) such that \( A_i \in \text{Executed}_\tau \) and \( \text{last}(\text{Executed}) \leq \text{time}(A_i) + y_i \), i.e., their execution windows include \( \text{last}(\text{Executed}) \). Finally, Let \( \mathcal{I}_C = \chi^0 \cup \chi^* \). An instantaneous reaction \( \mathcal{IR} \) for \( \text{env} \) is a decision

1. to execute all contingent time points in \( \mathcal{I}_C \) at time \( \text{last}(\text{Executed}) \), or
2. to assign to each proposition \( p \) a truth value if \( (P?, \text{last}(\text{Executed})) \in \text{Executed} \), 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 \( \text{env} \) 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 \( \Delta_{\text{env}} \) and \( \Delta_{\text{ctrl}} \) 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 \( \mathcal{PS} = (\text{Executed, Assigned}) \) be a locally consistent partial schedule; \( \Delta_{\text{ctrl}} \) be an RTED for \( \text{ctrl} \) and \( \Delta_{\text{env}} \) be an RTED for \( \text{env} \). We model the partial outcome of \( \Delta_{\text{ctrl}} \) and \( \Delta_{\text{env}} \) as a mapping \( \mathcal{PO} : \mathcal{PS}^* \times \Delta_{\text{ctrl}}^* \times \Delta_{\text{env}}^* \rightarrow \mathcal{PS}^* \) neglecting any instantaneous reaction. We have four possible cases:

1. \( \mathcal{PO}(\mathcal{PS}, \text{wait}, (t, \text{Contingent})) = \text{Executed} \cup \{(C, t) \mid C \in \text{Contingent}\} \).
2. \( \mathcal{PO}(\mathcal{PS}, (t_1, \text{NonContingent}), (t_2, \text{Contingent})) = \text{Executed} \cup \{(C, t_2) \mid C \in \text{Contingent}\} \) if \( t_2 < t_1 \).
3. \( \mathcal{PO}(\mathcal{PS}, (t, \text{NonContingent}), \text{wait}) = \text{Executed} \cup \{(X, t) \mid X \in \text{NonContingent}\} \).
4. \( \mathcal{PO}(\mathcal{PS}, (t_1, \text{NonContingent}), (t_2, \text{Contingent})) = \text{Executed} \cup \{(X, t_1) \mid X \in \text{NonContingent}\} \) if \( t_1 \leq t_2 \).
that in cases (3) and (4) respectively.

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

\[
\begin{align*}
\mathcal{PS}_0 &= (\emptyset, \emptyset) \\
\mathcal{PS}_{i+1} &= FO^1(\mathcal{PS}_i, \sigma_{ctrl}, \sigma_{env})
\end{align*}
\]

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

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

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 \((S, \rightarrow)\), 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.


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 [2]). A Timed Automaton (TA) is a tuple \(\langle \text{Loc}, \text{Act}, \mathcal{X}, \rightarrow, \text{Inv} \rangle\), where

- \(\text{Loc}\) is a finite set of locations (with \(L_0\) set as the initial one).
- \(\text{Act}\) 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.
- \(\rightarrow \subseteq \text{Loc} \times \mathcal{H}(\mathcal{X}) \times \text{Act} \times 2^\mathcal{X} \times \text{Loc}\) 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 \(x, y \in \mathcal{X}\), \(k \in \mathbb{N}^4\) and \(\sim \in \{<, \leq, =, >, \geq\}\). 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.
- \(\text{Inv}: \text{Loc} \to \mathcal{H}(\mathcal{X})\) 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 \(cX\) \(\leq 3\), then it has to leave the location. The pass transition can be taken whenever \(cX \geq 1\) and along with \(\text{Inv}(L_0)\) ensures that the TA will enter \(L_1\) at any instant such that \(1 \leq cX \leq 3\). When pass is taken, \(cX\) resets to 0. After that, the gain transition may be taken as soon as \(cX \geq 5\). 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 [37]). 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.

\[^{4}\text{The original definition allows one to consider the } \mathbb{Q} \text{ set as constant domain. However, in [2] the same authors show that it is possible to consider } \mathbb{N} \text{ without loss of generality.}\]
Figure 3: Examples of a) TA and b) TGA.

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 $\hat{c}, c_{\delta}, c_A$ and $c_C$. Solid arrows represent controllable transitions, whereas dashed arrows uncontrollable ones. The initial location is $L_0$, and goal is the location that $\text{ctrl}$ 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 $\langle L_1, c_A = \hat{c}, \text{ExA}, \{c_A\}, L_1 \rangle$ is taken resulting in the reset of $c_A$. After that, the pass transition is taken always at time 5. Therefore, the current location becomes $L_0$ and the clock $c_{\delta}$ is reset to 0. At time 6, both the ExC and gain transitions are enabled and $\text{ctrl}$ decides to take gain. At the same time, env decides to take ExC. Since ExC has priority, it executes first and, therefore, the clock $c_C$ is reset to 0. Then, the gain 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, $c_C$ will never be reset, preventing the controller to take the win transition.

In what follows, we provide an improvement of the encoding given in [13, 14] from a CSTNU into a suitable TGA such that the CSTNU is DC if and only if $\text{ctrl}$ 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, $\hat{c}$ is a clock representing the global time and it is never reset, whereas $c_{\delta}$ 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 \texttt{ctrl} can take transitions for executing non-contingent time points.

• \texttt{goal} is the final location that, if reached (by \texttt{ctrl}), implies DC of the initial CSTNU.

We model the interplay between \texttt{ctrl} and \texttt{env} with a clock \( c_{\delta} \) and two transitions:

1. \((L_1, \top, \text{pass}, \{c_{\delta}\}, L_0)\)
2. \((L_0, c_{\delta} > 0, \text{gain}, 0, L_1)\)

The first transition guarantees that, after some action(s) of \texttt{ctrl}, the environment can react immediately (\( \top \) shortens the absence of any clock constraint and it is always interpreted 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 \( c_{\delta} > 0 \)).

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

We associate a clock \( cX \) to each time point \( X \in \mathcal{T} \). If \( cX = \hat{c} \), it means that \( X \) has not been executed yet, whereas if \( cX < \hat{c} \), it means that \( X \) was executed at time \( \hat{c} - cX \). 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 \textit{uncontrollable} self-loop transition at location \( L_1 \)

\[(L_1, cX = \hat{c}, \text{Ex}X, \{cX\}, L_1)\]

This transition can be executed only when \( cX \) is equal to \( \hat{c} \) (i.e., \( X \) has not been executed yet). The transition resets \( cX \) 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 \([52]\) 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 \text{Ex}A_1 by appending \( bD = \hat{c} \land cD < \hat{c} \land cD > 0 \) to model that \( A_1 \) must be executed if only if \( d = \top \) (i.e., \( bD = \hat{c} \)), which also implies that \( A_1 \) must be executed after \( D? \) (i.e., \( D? \) have been executed \( cD < \hat{c} \)) and a positive amount of time \( \epsilon \) has elapsed \( (cD > 0) \).

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

\[
L_{\text{enc}}(X) : \bigwedge_{p \in L(X)} (bP = \hat{c} \land cP < \hat{c} \land cP > 0) \bigwedge_{\neg q \in L(X)} (bQ < \hat{c} \land cQ < \hat{c} \land cQ > 0)
\]

We now focus on constraints. Consider the requirement link \( D?[5,10], d \rightarrow A_1 \) 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 \text{Ex}A_1 by further appending \( cD < \hat{c} \land cD \geq 5 \land cD \leq 10 \), where the new conjuncts say that \( D? \) has already been executed \( (cD < \hat{c}) \) and \( A_1 - D? \in [5,10] \ (cD \geq 5 \land cD \leq 10) \). Since clocks are reset upon the execution of time points we just need to measure the elapsed time since the reset of \( cD \) to make sure that once we reset \( cA_1 \) the constraint holds (see also the modeling of contingent links that we discuss below). More formally:
Figure 4: TGA encoding the CSTNU in Figure 2. $L_0$ is the initial location, $L_1$, $L_2$, $L_3$, $L_4$, goal are urgent. Solid (resp., dashed) edges model controllable (resp., uncontrollable) transitions.
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 \( x > 0 \). \( \Pi : \mathcal{T} \rightarrow 2^\mathcal{T} \) returns the predecessors of a time point and it is formalized as

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

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

\[
\Pi_{enc}(Y) = \bigwedge_{X \in \Pi(Y)} cX < \hat{c} \land cX \geq x \land cX \leq y,
\]

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

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

\[
(L_1, cX = \hat{c} \land L_{enc}(X) \land \Pi_{enc}(X), \text{Ex}X, \{cX\}, L_1)
\]

Figure 4 contains 7 transitions (\( \text{ExD, ExA}_1, \text{ExA}_2, \text{ExA}_3, \text{ExA}_4 \) and \( \text{ExA}_5 \)) modeling the execution (by \( \text{ctrl} \)) 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 [52].

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

\[
(L_0, cP < \hat{c} \land cP = 0 \land bP = \hat{c}, \text{pFalse}, \{bP, c_\delta\}, L_0)
\]

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

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

\[
(L_0, cA < \hat{c} \land cC = \hat{c} \land cA \geq x \land cA \leq y, \text{ExC}, \{cC, c_\delta\}, L_0)
\]

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

\[
(L_1, cA < \hat{c} \land cC = \hat{c} \land cA > y, \text{failC}, \emptyset, \text{goal})
\]

once it gets back to \( L_1 \). Indeed, this last transition says that the activation time point has been executed (\( cA < \hat{c} \)), the contingent time point has not be set (\( cC = \hat{c} \)) and it is currently out of its range of allowed values (\( cA > y \)). 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 \textit{fail} transition. In general, fail transitions for contingent links are always necessary because, otherwise, \texttt{env} would always be able to prevent \texttt{ctrl} from winning the game just “waiting forever” in \texttt{L0}.

In Figure 4, there are 5 transitions for the execution of \texttt{C1}, \texttt{C2}, \texttt{C3}, \texttt{C4} and \texttt{C5} (\texttt{ExC1}, \texttt{ExC2}, \texttt{ExC3}, \texttt{ExC4}, \texttt{ExC5}) and 5 to handle their failure (\texttt{failC1}, \texttt{failC2}, \texttt{failC3}, \texttt{failC4}, \texttt{failC5}).

We model the winning conditions of the game as a winning path of \( n + 1 \) urgent locations \( L_\Box \rightsquigarrow L_{\ell_1} \rightsquigarrow \cdots \rightsquigarrow L_{\ell_n}(= \texttt{goal}) \) going from \texttt{L0} to \texttt{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 \( L_\Box \), reachable as soon as \( c_\delta > 0 \) 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 \( \ell_i \), then all time points labeled by \( \ell_i \) must have been executed and all constraints labeled by \( \ell_i \) 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 \( L_{\ell_i} \) [13].

Therefore, each set of transitions between two consecutive locations represents two cases:

1. if \( \ell_\text{cps} \not\ni \ell_i \), then at least one transition of the set allows \texttt{ctrl} to move to the next location, or

2. if \( \ell_\text{cps} \ni \ell_i \), then exactly one transition allows \texttt{ctrl} to move to the next location if and only if all time points labeled by \( \ell_i \) have been executed and all the constraints labeled by \( \ell_i \) have been satisfied.

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

The conditional meta constraint for generating such a set of transitions is: “If \( \mathcal{D} \)? 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:

\[
(cD < \hat{c} \land bD = \hat{c}) \Rightarrow (cA_1 < \hat{c} \land cC_1 < \hat{c} \land cA_1 - cD \leq -5 \land cD - cA_1 \leq 10 \land \\
cA_2 - cC_1 \leq -5 \land cC_1 - cA_2 \leq 10)
\]

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

\[
\neg(cD < \hat{c} \land bD = \hat{c}) \lor \\
(cA_1 < \hat{c} \land cC_1 < \hat{c} \land cA_1 - cD \leq -5 \land cD - cA_1 \leq 10 \land cA_2 - cC_1 \leq -5 \land cC_1 - cA_2 \leq 10)
\]

\footnote{For each \( l \leq Y - X \leq u \in \mathcal{C} \), the corresponding guard is \( l \leq (\hat{c} - cy) - (\hat{c} - cx) \leq u \), which simplifies in \( cx - cy \geq l \land cx - cy \leq u \) as \( \forall X \in \mathcal{T} \). \((\hat{c} - cx)\) is the time in which \( X \) was executed.}
which simplifies to
\[
(cD = \hat{c}) \lor \\
(bD < \hat{c}) \lor \\
(cA_1 < \hat{c} \land cC_1 < \hat{c} \land cA_1 - cD \leq -5 \land cD - cA_1 \leq 10 \land cA_2 - cC_1 \leq -5 \land cC_1 - cA_2 \leq 10)
\]

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

Figure 4 represents these three disjuncts as a set of 3 transitions: \texttt{sat}_d\ whose guard is \((cA_1 < \hat{c} \land cC_1 < \hat{c} \land cA_1 - cD \leq -5 \land cD - cA_1 \leq 10 \land cA_2 - cC_1 \leq -5 \land cC_1 - cA_2 \leq 10)\), \texttt{skip}_1\ whose guard is \(cD = \hat{c}\) and \texttt{skip}_2\ whose guard is \(bD < \hat{c}\). We generate the set of transitions going from \(L_\square\) 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 [13, 14] 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 \texttt{env} to always prevent \texttt{ctrl} from reaching \texttt{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 \textit{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, \textit{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 \textit{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) $\tau$ is a (temporal) assertion defined according to the grammar:

$$\tau ::= \oslash | \theta k | \theta X + k | \tau_1 \land \tau_2$$

where $\oslash$ is the empty constraint, $\theta \in \{>,<,\geq,\leq,=\}$, $k \in \mathbb{N}$, $X$ is a time point, and $\tau_1, \tau_2$ are two TEs. There are 4 types of TEs:

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

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

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

**Type 3**: $\tau = \tau_1 \land \tau_2$ (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 $\tau$, with respect to a particular instant $t$, by means of the following interpretation.

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

1. $t \models \oslash$
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. $t \models \tau_1 \land \tau_2$ 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 $(\mathcal{RA}, \mathcal{RE})$, where:

- $\mathcal{RA} \subseteq \mathcal{R} \times \mathcal{T}$ determines which resources can be committed for each time point. We write $\mathcal{R}(X) = \{r \mid (r,X) \in \mathcal{RA}\}$ 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 $X \in \mathcal{T}$, $\mathcal{R}(X) \neq \emptyset$.

- $\mathcal{RE}: \mathcal{RA} \rightarrow \mathcal{TE}$ 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 $t \models \mathcal{RE}(r,X)$, 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 \((r,X) \in RA \land RE(r,X) = \tau \). 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 \( \Box \) 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 \( <X+15 \rangle \) 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 \( \tau \) 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 \( \{(r_1,r_2) \mid r_1,r_2 \in \mathcal{R} \land r_1 \neq r_2\} \), 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 \( \tau \) 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,

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

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 $\langle \rightarrow X+10,= \rightarrow Y \rangle$, 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
$\rho$. In this case, it applies only to $r_2$ because $\rho$ 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:

$$
\mathcal{R}(Y) \text{ when } t<1 \\
\{r_1(\circ), r_2(<X+15)\} \sim \{r_1(\circ), r_2(>11 \land <16)\}
$$

The delay allowed for executing $Y$ after $X$ is $[5, 20]$. If $\text{ctrl}$ 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 CSTNU 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

$$
\langle \mathcal{T}, \mathcal{P}, L, \mathcal{O}T, O, C, \mathcal{L}, \mathcal{R}, RA, RE, RRC \rangle,
$$

where:

1. $\langle \mathcal{T}, \mathcal{P}, L, \mathcal{O}T, O, C, \mathcal{L} \rangle$ is a CSTNU.
2. $\mathcal{R} = \{r_0, r_1, \ldots \}$ 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. $C_2 \leftarrow C_1 \leftarrow A_5 \leftarrow A_4 \leftarrow A_3 \leftarrow A_2$ (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). $A_3 \leftarrow A_2 \leftarrow A_4$ (RRC$_2$) and
$A_4 \leftarrow A_3 \leftarrow A_2 \leftarrow A_1$ (RRC$_3$) model a “no multi-tasking” policy for 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.

20
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 CSTNU, 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 \(\mathcal{PS} = (\text{Executed}, \text{Assigned})\) 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 \(X \in \text{Executed}_T\), 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 \((r, X, t) \in \text{Executed}_T\), \(r \in R(X)\) and \(t \models RE(r, X)\), where \(RE(r, X)\) is the temporal expression associated to the pair \((r, X)\). The set of all possible partial schedules remains

\[\{1440, 2160\}, \Box\]

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

---

Footnotes:

6 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
represented by $\mathcal{PS}^*$. 

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

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

- $\Delta_{\text{ctrl}} = \text{wait}$ is the same as that given in **Definition 3**.
- $\Delta_{\text{ctrl}} = (t, \text{NonContingent})$ represents the conditional constraint: “if $\text{env}$ does nothing before time $t$, then for each pair $(r, X) \in \text{NonContingent}$, commit the resource $r$ to execute time point $X$ at time $t$.” Such a decision is applicable if and only if $t > \text{last}(\text{Executed})$, $\text{NonContingent}$ 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 committal at time $t$ and $X_i$ is a non-contingent unexecuted time point such that $t > \text{last}(\text{Executed})$.

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

- $\Delta_{\text{env}} = \text{wait}$ is the same as that given in **Definition 4**.
- $\Delta_{\text{env}} = (t, \text{Contingent})$ is the same as that given in **Definition 4** committing $\text{res}(A)$ to execute $C$. In other words, such a decision is applicable if and only if $t > \text{last}(\text{Executed})$, $\text{Contingent}$ 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 $A \in \text{Executed}^\tau$, $r = \text{res}(A)$ is the resource that was committed for $A$ and $t \in [\text{time}(A) + x, \text{time}(A) + y]$.

$\Delta^*_{\text{ctrl}}$ and $\Delta^*_{\text{env}}$ still denote the sets of all RTEDs for $\text{ctrl}$ and $\text{env}$. 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 $\Delta_{\text{ctrl}}$ and $\Delta_{\text{env}}$.

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

1. $PO(\text{Executed}, \text{wait}, (t, \text{Contingent})) = \text{Executed} \cup \{(\text{res}(A), C, t) \ | \ C \in \text{Contingent}\}$. Also, for any $(C, \tau, Y, \rho) \in \mathcal{RRC}$ such that $Y \notin \text{Executed}^\tau$ we have that $\forall r_Y \in \mathcal{R}(Y), (\text{res}(A), r_Y) \in \rho \implies \text{RE}(r_Y, Y) := \text{RE}(r_Y, Y) \land \tau$.

2. $PO(\text{Executed}, (t_1, \text{NonContingent}), (t_2, \text{Contingent})) = \text{Executed} \cup \{(\text{res}(A), C, t_2) \ | \ C \in \text{Contingent}\}$ if $t_2 < t_1$. Also, for any $(C, \tau, Y, \rho) \in \mathcal{RRC}$ such that $Y \notin \text{Executed}^\tau$ we have that $\forall r_Y \in \mathcal{R}(Y), (\text{res}(A), r_Y) \in \rho \implies \text{RE}(r_Y, Y) := \text{RE}(r_Y, Y) \land \tau$.
(3) \( PO(\text{Executed}, (t, \text{NonContingent}), \text{wait}) = \text{Executed} \cup \{ (r_i, X_i, t) \mid (r_i, X_i) \in \text{NonContingent}, \forall i = 1, \ldots, k \} \). Also, every time we add \((r_i, X_i, t)\) to \text{Executed} we fire the related RRCs (if any). That is, for any \( \langle X_i, \tau, Y, \rho \rangle \in \mathcal{RRC} \) such that \( Y \notin \text{Executed}_\tau \) we have that \( \forall r_Y \in \mathcal{R}(Y), (r_i, r_Y) \in \rho \implies \text{RE}(r_Y, Y) := \text{RE}(r_Y, Y) \land \tau \).

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

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 \text{ctrl} and \text{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 \( \sigma_{\text{ctrl}} \) such that for all RTED-based strategies \( \sigma_{\text{env}} \), the variable assignments \( (r, X, k) \) in the complete schedule \( FO^*(\sigma_{\text{ctrl}}, \sigma_{\text{env}}) \) satisfy all temporal constraints of the underlying CSTNU, and each assignment \( (r, X, k) \) satisfies both \( r \in \mathcal{R}(X) \) and \( k \models \text{RE}(r, X) \).

## 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 \text{ctrl} and \text{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 \text{RA} relation. Therefore, to model which resource has been committed for which time point, we start by associating a dedicated clock \( rX \) to each element \( (r, X) \in \mathcal{R} \) and interpreting the value of such clocks as follows (\( cX \) is the clock associated to \( X \)):

1. If \( rX = cX = \hat{c} \), it means that \( X \) has not been executed yet and \( r \) is committable for \( X \).
2. If \( rX = cX < \hat{c} \), it means that \( X \) has been executed and \( r \) is not the resource committed for \( X \).
3. If \( rX = \hat{c} > cX \), it means \( r \) was committed for \( X \) at time \( \hat{c} - cX \).
4. If \( rX < cX = \hat{c} \), it means that \( X \) has not been executed yet and \( r \) cannot be committed for \( X \) because \( r \) is not available. In details, \( rX \) becomes less than \( cX \) by a reset action. The reset of \( rX \) occurs when it is necessary to prevent \( r \) from being committed for \( X \) since \( \hat{c} \not\models \text{RE}(r, X) \).
Differently from what we have done for clocks associated to time points, here $rX$ clocks may be reset more than once. If $r$ is committable for $X$ and $rX$ 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 $cX < \hat{c}$ and find the unique clock $rX$ such that $rX > cX$. It is guaranteed that all other clocks $r_jX$ where $r_j \in \mathcal{R}(X)$ and $r_j \neq r$ are equal to $cX$.

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

- $wD$ models $\mathcal{R}(D?) = \{wf\}$
- $aA_1, bA_1$ model $\mathcal{R}(A_1) = \{Alice, Bob\}$
- $aC_1, bC_1$ model $\mathcal{R}(C_1) = \{Alice, Bob\}$
- $1A_2, mA_2$ model $\mathcal{R}(A_2) = \{Lila, Mike\}$
- $1C_2, mC_2$ model $\mathcal{R}(C_2) = \{Lila, Mike\}$
- $eA_3, tA_3$ model $\mathcal{R}(A_3) = \{Evie, Tim\}$
- $eC_3, tC_3$ model $\mathcal{R}(C_3) = \{Evie, Tim\}$
- $eA_4, tA_4$ model $\mathcal{R}(A_4) = \{Evie, Tim\}$
- $eC_4, tC_4$ model $\mathcal{R}(C_4) = \{Evie, Tim\}$
- $1A_5, mA_5$ model $\mathcal{R}(A_5) = \{Lila, Mike\}$
- $1C_5, mC_5$ model $\mathcal{R}(C_5) = \{Lila, Mike\}$

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 $\vec{c} = (\hat{c}, c_5, cX, \ldots, cY, bP, \ldots, r_0X, \ldots, r_0Y, \ldots)$ 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:

- $L = L_0$, $c_5 = 0$, last(Executed) = $\hat{c}$.

- For each executed time point $X$, $time(X) = \hat{c} - cX$ and $res(X) = rX$, where $rX$ is the only $r_iX$ clock not reset.

- For each unexecuted time point $X$, $time(X) = \hat{c}$ and $res(X)$ is not defined (i.e., all $r_iX = \hat{c}$).

- For each executed observation time point $P?$, if $p = \top$ then $bP = \hat{c}$, and if $p = \bot$ then $bP < \hat{c}$. If the time point is not executed, the value of $bP$ 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
(a) Circular path modeling the execution of \( Y \). A run enters this circular path only if \( Y \) is unexecuted (\( L_1 \rightarrow Y \)). Then, it verifies all the RRCs targeting \( Y \) (if any) (\( Y \rightarrow \cdots \rightarrow Y_n \)). After that, it commits one and only one committable resource among those associated to \( Y \) (if any) (\( Y_n \leftarrow Y_r \)). Finally, it fixes the execution time of \( Y \) going back to \( L_1 \) (\( Y_r \leftarrow L_1 \)). 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 \( Y \) 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 (\( Y_i \rightarrow F \)), otherwise it skips the verification (\( Y_i \rightarrow Y_j \)). At \( F \), either \( \theta k \) is satisfied, and then no resource is blocked (\( F \rightarrow Y_j \)), or \( \theta k \) is violated (\( F \rightarrow V \)), and a few resources may eventually be blocked (\( V \rightarrow Y_j \)).

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

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.
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 \(rY\).

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 \(rY = \hat{c}\). 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 \(rY\). 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 \(r_iY = \hat{c}\) as the unique clock constraint in its guard) and resets all clocks \(r_jY\) 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 \(cY\). Eventually, if \(r\) is committed for \(Y\) at time \(\hat{c} - cY\), the following three conditions hold:

1. \(r \in \mathcal{R}(Y)\)
2. \(\hat{c} - cY \models RE(r, Y)\)
3. \(rY > cY\) and \(r_iY = cY\) 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:

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

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 \( \hat{c} \) 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

\[
\langle X, >\leq_1 3 \land >\leq_2 C \land >\leq_3 Z + 5, Y, \rho \rangle
\]

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

\[
\{(X, > 3 \land \leq 10, Y, \rho), (X, > C, Y, \rho), (X, \leq Z + 5, Y, \rho)\}
\]

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 \( \hat{c} \theta_k \) modeling the clock constraint \( \theta_k \) for \( X \). From Definition 16 it holds that \( t \models \theta_k \) iff \( t \theta_k \) whatever \( \theta \) is. Since in a CSTNUR TEs are evaluated with respect to global time (modeled by \( \hat{c} \)), it is sufficient to substitute \( \hat{c} \) for \( t \) getting \( \hat{c} \theta_k \).

In the second case, \( t \not\models \theta_k \) and thus we must block all resources having \( \theta_k \) associated. To achieve this aim, the run first enters \( V \). Either one or two transitions connect \( F \) to \( V \) according to which \( \theta \) is specified. If \( \theta \) is ‘=’ (i.e., the corresponding clock constraint is \( \hat{c} = k \)), then there are two transitions, one having guard \( \hat{c} < k \) and the other having guard \( \hat{c} > k \). 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 \( \neg(\hat{c} \theta_k) \). In Figure 7c such transitions have labels \text{viol}1 and \text{viol}2, where \text{viol}2 (drawn in gray) exists if and only if \( \theta \) is ‘=’. \text{\( \bowtie_1 \) and \( \bowtie_2 \) model the new \( \theta \) operators arising from the negation of \( \theta_k \). Finally, a set of transitions connects \( V \) to \( Y_j \). There exists one transition for each resource \( r_X \in R(X) \) 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 \( \rho \)
expressed in the considered RRC must be blocked by resetting their associated clocks. That is, the guard of each transition block is \( rX > cX \) (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:

\[
R_i = \{ r_j Y \mid r_j \in R(Y) \land (r_i, r_j) \in \rho \}
\]

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 \( (X, X_i \theta k_i, Y, \rho) \). Figure 7d shows the encoding of such an RRC, where rather than encoding \( (X, X_i \theta k_i, Y, \rho) \) and obtaining \( (X, \theta k_i Y, \rho), \ldots, (X, \theta n k_n Y, \rho) \) and then encoding each RRC according to Figure 7c resulting in a circular path \( Y_1 \rightarrow \cdots \rightarrow Y_n \rightarrow Y_r \) of \( n \) DAGs, we generate a refined shorter path \( Y_1 \rightarrow \cdots \rightarrow Y_r \), consisting of one DAG only, able to deal with the entire conjunction of TEs. This encoding substantially extends the \( \text{sat} \) and \( \text{viol} \) transitions in Figure 7c. All other transitions remain the same. We refine \( \text{sat} \) so that it verifies the clock constraint \( \overline{c \theta k_1} \land \cdots \land \overline{c \theta n k_n} \).

To generate \( \text{viol} \) transitions, we proceed exactly as we did in Figure 7c. That is, we compute \( \neg(\overline{c \theta k_j} \land \cdots \land \overline{c \theta n k_n}) \) resulting in \( \bigvee_j \neg(\overline{c \theta k_j}) \), where, again, if \( \theta \) 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: \( (X, =6, Y, \rho) \). \( =6 \) is translated as clock constraint \( \hat{c} = 6 \) (\( \text{sat} \)), and \( \neg(\hat{c} = 6) \) becomes \( (\hat{c} < 6) \lor (\hat{c} > 6) \), from which \( \text{viol}_1 \) and \( \text{viol}_2 \) are generated (connecting \( F \) to \( V \)).

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

### 5.2.2 Encoding an RRC containing a TE of Type 2

We encode an RRC having the form \( (X, \theta Z + k, Y, \rho) \) 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 \( \theta \). 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 \( (cZ < \hat{c}) \), whereas it moves to \( Z_u \) if and only if \( Z \) has not \( (cZ = \hat{c}) \).

At \( Z_u \) we might block some resource(s) according to \( \theta \). 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 \( R(X) \). 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 \( \hat{c} \theta k \) \((\hat{c} \gg Z + k)\), which simplifies to \( cZ \theta k \). If \( cZ \theta k \) is true, no resource will be blocked and a \( \text{sat} \) transition allows the run to move to \( Y_j \). If \( cZ \theta k \) does not hold, then \( \neg(cZ \theta k) = cZ \not\gg k \) and, therefore, there are one or two \( \text{viol} \) 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 $(X, \theta Z + k, Y, \rho)$. 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 viol1_2 transitions.

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),
Figure 9: Circular paths modeling the authorized execution of the non-contingent time points of the CSTNUR in Figure 6 and two RRC encodings.
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 \(|R(A)|\) transitions connects \(C\) to \(C_r\). A run can enter \(C\) as soon as the clock constraint for contingent time points (i.e., \(cA < \hat{c} \land cC = \hat{c} \land cA \geq x \land cA \leq y\)) 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 \(r_iExC\) going from \(C\) to \(C_r\). The only enabled transition \(r_iExC\) is the one whose guard contains the clock constraint \(r_iC > cA\), i.e., the transition associated to the resource \(r_i\) committed for \(A\). Indeed, for all other \(r_j \in R(A)\) where \(r_j \neq r_i\), it holds that \(r_jC \leq cA\). Finally, the run moves back to \(L_0\) by resetting \(cC\), i.e., fixing the execution time for \(C\) and \(c_δ\).

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 \(\mathcal{N} = \langle T, P, L, OT, O, C, L, R, RA, RE, RRC \rangle\), for each RRC \(r \in RRC\), the complexity for determining its corresponding TGA fragment (see Section 5) is \(O(|R|^2)\), where \(R\) 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 \(|R|\). Moreover, each TGA block edge \(block_i\) has to reset clocks associated to some resources. Indeed, in the worst case, its label may have length \(O(|R|)\) 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 \(O(|R|^2)\).

**Lemma 2** (Resource Commitments Encoding Complexity). Given a CSTNUR \(\mathcal{N} = \langle T, P, L, OT, O, C, L, R, RA, RE, RRC \rangle\), the complexity for determining
(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 $C \rightarrow C_r$ will be enabled. For each of these transitions $R_i = \{r_j \mid r_j \in R(C) \land r_j \neq r_i \}$.

(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.
all TGA fragments corresponding to the resource commitments (as, for example, all fragments in Figure 9a) is $O(n^3)$, where $n$ is the encoding length of $N$.

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

- the edge $L_1 \rightarrow Y$, 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 $O(|\mathcal{P}|)$ clocks specifications, where $\mathcal{P}$ is the set of possible propositions of the network. Therefore, its building complexity is $O(|\mathcal{P}|)$.

- each fragment $Y_{i-1} \rightarrow Y_i$, because it represents a DAG handling the validation of the $i$th RRC associated to $Y$. The complexity of each fragment is $O(|R|^2)$ as proved in Lemma 1, and

- the edges $r_i \mathcal{E} Y_i, i = 1, \ldots, m$ from $Y_n$ to $Y_r$, because they can be $|R|$ at most and each of them can have label with length $O(|R|)$. 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 $O(|R|^2)$.

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

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

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 $C$ and $C_r$ there are $|R|$ 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 $|R| - 1$ clock resets.

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

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

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 $O(n^3)$, where $n$ is the encoding length of $N$.

The complexity for encoding the winning path is linear in the number of distinct labels present in the input network [14]). The number of distinct labels in $N$ is $O(n)$. Therefore, the overall complexity for building a TGA from a CSTNUR $N$ is $O(n^3)$.

We prove the correctness of the encoding given in Section 5 by means of the following two theorems. Such theorems extend those given in [14] 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 $\text{ctrl}$ synthesized by reachability analysis of the resulting TGA corresponds to an RTED-based strategy (Theorem 3).

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

- $L = L_0$, $c_\delta = 0$, $\text{last}($Executed$) = \hat{c}$.
- For each executed time point $X$, $\text{time}(X) = \hat{c} - cX$ and $\text{res}(X) = rX$, where $rX$ is the only $r_iX$ clock not reset.
- For each unexecuted time point $X$, $\text{time}(X) = \hat{c}$ and $\text{res}(X)$ is not defined (i.e., all $r_iX = \hat{c}$).
- For each executed observation time point $P?$, if $p = \top$ then $bP = \hat{c}$, and if $p = \bot$ then $bP < \hat{c}$. If the time point is not executed, the value of $bP$ 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 $c_\delta = 0$, 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 $\text{env}$, in which a set of one or more
contingent time points and/or proposition assignments can be executed simultaneously. Suppose
that $\text{env}$ does not take any transition when $c_\delta = 0$. As soon as $c_\delta > 0$, both $\text{ctrl}$ and $\text{env}$
may execute enabled transitions (i.e., those with true guards). For example, $\text{env}$ might decide to
execute one or more contingent time-points $C_1, \ldots, C_n$ when $c_\delta = 3$. That would correspond to
$\Delta_{\text{env}} = (k, \{C_1, \ldots, C_n\})$, where $k = \text{last} (\text{Executed}) + 3$.

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

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

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

Finally, if at time $\text{last}(\text{Executed})$, $\text{ctrl}$ and $\text{env}$ both decide to execute some time points at time
$\text{last}(\text{Executed}) + 1$, then the CSTNUR semantics (inheriting the CSTNU semantics) ensures
that $\text{ctrl}$ time points are executed first, and that $\text{env}$ is able to instantaneously react if it decides to
do so (equivalent to $\text{ctrl}$ transitions having priority over $\text{env}$). As soon as the execution returns to
the location of $\text{env}$, $\hat{c}$ has still the same value $\text{last}(\text{Executed}) + 1$ 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.

\begin{theorem}
Let $\mathcal{N}$ be a CSTNUR, $\mathcal{G}$ be the encoding of $\mathcal{N}$, and $\sigma_{\text{ctrl}}$ be a winning TGA counter-
strategy for $\text{ctrl}$. Then, there is an equivalent RTED-based strategy $\sigma_{\text{ctrl}}$ for $\text{ctrl}$ that will ensure

\end{theorem}
the satisfaction of all temporal constraints in \( \mathcal{N} \) and all RRCs, if fired, whatever the contingent durations and truth value assignments.

**Proof.** If \( \mathcal{N}, \mathcal{G}, \sigma_0 \) are as assumed, then we can interpret \( \sigma_0 \) as a function \( \sigma_0 : S \rightarrow Act \cup \text{wait} \), where \( S \) is the state set of the TGA and \( Act \) is the \texttt{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 \( cX \), it holds that:

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

For each observation time point \( P' \), when \( cP = \hat{c} \), the associated proposition modeled by \( bP \) is \( bP = \hat{c} \) (i.e., unknown before \( P' \) executes), and when \( cP < \hat{c} \), either \( bP = \hat{c} \) (i.e., \( \top \)) or \( bP < \hat{c} \) (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 \( \hat{c} > \text{last(Executed)} \), i.e., that some positive time has elapsed since the last execution event in \( \mathcal{P}S \). If nothing has happened, it means that there has been a sequence of \texttt{gain} and \texttt{pass} transitions going back and forth between \texttt{env} and \texttt{ctrl} locations. In such a loop, \texttt{ctrl} has not executed any non-contingent time point, and \texttt{env} has just waited. Let \( (L_0, \vec{c}) \) be a state preceding such loop. Then, for some \( \epsilon > 0 \), all the clocks in \( \vec{c} \) equal those in \( \vec{c} + \epsilon \), and by construction, \( \text{last(Executed)} \) refers to the clocks in \( \vec{c} \). Next, let \( d = \min\{d \mid \sigma_0(L_0, \vec{c} + d) \neq \text{wait} \land \sigma_0(L_0, \vec{c} + d) \neq \text{pass} \} \) be the minimum time that can elapse from \( \vec{c} \) before the strategy \( \sigma_0 \) recommends a transition different from \texttt{gain} and \texttt{pass}, and let \( \vec{c}_0 = \vec{c} + d \). The unique sequence of execution transitions at \texttt{ctrl} is \( \sigma_0(L_1, \vec{c}_0), \ldots, \sigma_0(L_1, \vec{c}_n) \), where each \( \vec{c}_{i+1} = \vec{c}_i \), except for \( cX \) with \( X \) the time point executed by \( \sigma_0(L_1, \vec{c}_i) \). The termination of this sequence of transitions is guaranteed since time points are finite and can only be executed once. If \( \sigma_0(L_1, \vec{c}_n) \) is the last execution transition, then \( \text{pass} = \sigma_0(L_1, \vec{c}_n) \). 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, \( \sigma_0(L_1, \vec{c}_0), \ldots, \sigma_0(L_1, \vec{c}_n) \), are all 0 in \( \vec{c}_n \).

Next, let \( t \) be the time at which \( \sigma_0 \) recommends \texttt{ctrl} a non-trivial transition, and \texttt{NonContingent} be the set of pairs (resource, time point) corresponding to the execution transitions, \( \sigma_0(L_1, \vec{c}_0), \ldots, \sigma_0(L_1, \vec{c}_n) \). Then \( (t, \text{NonContingent}) \) is a \( \Delta_{\text{ctrl}} \) corresponding to what the strategy recommends at \( (L_0, \vec{c}) \). Note that \texttt{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, \texttt{env} may decide to intervene before time \( t \) arrives, by executing one or more contingent time-points and effectively generating a new partial schedule \( \mathcal{P}S' \). In that case, the same procedure could be applied to \( \mathcal{P}S' \) to generate an appropriate \( \Delta_{\text{ctrl}} \). Since the guard of \texttt{gain} requires a positive time delay, that \( \Delta_{\text{ctrl}} \) is properly prohibited from any kind of instantaneous reaction (by \texttt{ctrl}). This procedure gives a mapping from any \( (L_0, \vec{c}) \) state that is reachable following \( \sigma_0 \).

\[ \square \]

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 [5]. 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) [34] 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 Ask(c) (is c true?) or the label 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 [44], 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) [43], 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 [35], 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, [31]), 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 [22] 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 [22], the authors proved that Labeled STNs are equivalent to DTNs [43]. 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 [51], 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 CSTPs [44] 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 [36]. 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 [36], Pike takes as input (1) a TPNU, (2) the initial and goal states (sets of Planning Domain Definition Language (PDDL) predicates, [26]), (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 [32], the authors proposed Conditional Simple Temporal Networks (CSTNs) extending the CSTN implicitly proposed in [44] by labeling both time points and constraints by conjunctions of propositional letters, i.e., conditions. In [44] 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 [10].

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, in [30]. On the other side, CSTNURs cannot represent decisions as in TPNs, Drake and Pike. The CSTN [44, 32] 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 [31]. In [29], the authors 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 in [13, 14].

In [50], a Controllable Conditional Temporal Problem with Uncertainty (CCTPU) extends CCTP to address temporal plans with also uncertain durations. 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 [52, 54]. 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 as shown in [8].

Let us now informally discuss the expressivity of the CSTNUR formalism with respect to the CSTNU one. Even though it seems plausible that resource allocations can be modeled as decisions, a polynomial-size encoding seems to be unfeasible. 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 [14]. 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.
Figure 11: Deciding dynamic controllability of CSTNURs via TGAs: direct encoding (path above), reduction to CDTNUs (path below).

(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.

(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, \[14\]), 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 \[9\], 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 $p = q = \top$). 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 [9]. 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 [9]. 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).

**Figure 13: Fragment of CSTNUR**

**Definition 24.** [14] A **Conditional Disjunctive Temporal Network with Uncertainty (CDTNU)** is a tuple $(\mathcal{T}, \mathcal{C}, \mathcal{L}, \mathcal{OT}, \mathcal{O}, \mathcal{P}, \mathcal{L})$, where

- $\mathcal{T}, \mathcal{L}, \mathcal{OT}, \mathcal{O}, \mathcal{P}$ are the same as those given for a CSTNU (Definition 1).
- $\mathcal{C}$ is a set of constraints $(\phi, \ell)$, where $\phi$ 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 $\ell \in \mathcal{P}^*$.
- $\mathcal{L}$ is a set of contingent links $(A,B,C)$, where $A,C \in \mathcal{T}$ and $B$ 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 $(\phi, \ell)$ are such that $\phi$ does not contain any disjunction, then the CDTNU boils down to a classic CSTNU.

For example, if $\mathcal{C}$ contains $\phi = ((Y - X \leq -3) \lor (Y - X \geq 4), p \land q)$ 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 $(A, \{[2,3] \lor [5,7]\}, C)$ 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
\[ X = k, \ X \leq k, \text{ and } X \geq k \text{ as shorts for } X - Z = k, \ X - Z \leq k, \ Z - X \leq -k, \text{ where } Z \text{ is the zero time point.} \]

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 \( L(X_{r_1}) = \cdots = L(X_{r_n}) = \square \). 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

\[
(\underbrace{X_{r_1} = X \land \cdots \land X_{r_j} = h^*}_{r_1 \text{ is committed for } X}) \lor \cdots \lor (\underbrace{X_{r_1} = h^* \land \cdots \land X_{r_j} = X\}_{r_n \text{ is committed for } X}), \square
\]

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

- \( (Z_{r^*} = Z \lor Z_{r^*} = h^*, \square) \) (but this constraint really doesn’t matter)

- \( \underbrace{(A_R = A \land A_Q = h^*\}_{\phi_1} \lor (A_R = h^* \land A_Q = A), \square\)

- \( \underbrace{(Y_R = Y \land Y_Q = h^*\}_{\phi_2} \lor (Y_R = h^* \land Y_Q = Y), \square\)

To model the resource commitment for \( Z, A \text{ and } Y \).

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

\[
\text{Commit the same resource for } C
\]

\[
((A = A_{r_1} \land C = C_{r_1}) \lor \cdots \lor (A = A_{r_n} \land C = C_{r_n}), \square)
\]
In Figure 13 we have the contingent link \((A, 1, 2, C)\), therefore we add
\[
(A = A_R \land C = C_R) \lor (A = A_Q \land C = C_Q), \Box
\]

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 \((X, \tau, \rho, Y)\) we add a time point \(X_Y\) \((L(X_Y) = \Box)\) and add the disjunctive constraint \((X_Y = X \lor X_Y = h^*, \Box)\). If \(X_Y = X\), then \(X\) is executed before \(Y\) (even when \(Y - X = 0\)). If we have another RRC \((Y, \tau, \rho, X)\), then we will add a \(Y_X\) and \((Y_X = X \lor Y_X = h^*, \Box)\) and also \(((X_Y = X \land Y_X = h^*) \lor (X_Y = h^* \land Y_X = Y), \Box)\) (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 \((X, \tau, \rho, Y)\) we add the pair of constraints

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

Consider the RRC \(A \rightarrow Y\) labeled by \(< 3, \neq\) 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

- \((A_Y = A \lor A_Y = h^*, \Box)\)
- \((A_Y = A \Rightarrow A - Y \leq 0, \Box)\)
- \((A_Y = h^* \Rightarrow A - Y \geq 0, \Box)\)

Resources committed for \(A\) and \(Y\) are different

- \((A_Y = A \land ((A_R = A \land Y_Q = Y) \lor (A_Q = A \land Y_R = Y)) \Rightarrow Y \leq 3, \Box)\)

Consider RRC \(A \rightarrow Y\) labeled by \(\geq A + 2, =\) 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

Resources committed for \(A\) and \(Y\) are equal

- \((A_Y = A \land ((A_R = A \land Y_R = Y) \lor (A_Q = A \land Y_Q = Y)) \Rightarrow Y \geq A + 2, \Box)\)

Finally, consider the RRC \(A \rightarrow Y\) labeled by \(> C + 3, =\) 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

\[ \begin{align*}
\phi_{10} & : (C_Y = C \lor C_Y = h^*, \Box) \\
\phi_{11} & : (C_Y = C \Rightarrow C - Y \leq 0, \Box) \\
\phi_{12} & : (C_Y = h^* \Rightarrow C - Y \geq 0, \Box)
\end{align*} \]

Resources committed for A and C are equal

\[ (\phi_{13}, \ell) \text{ is } ((A_Y = A \land (A_R = A \land Y_R = Y) \lor (A_Q = A \land Y_Q = Y))) \Rightarrow ((C_Y = C \Rightarrow Y - C \leq 3) \land (C_Y = h^* \Rightarrow Y - Y \leq -1)), \Box \]

First case

No sol. exists

Second case

Note that “no solution exists” is modeled as a negative self loop. In this example, we used \(-1\) as weight for \(Y \rightarrow Y\), 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 \(\theta\).

Finally, \(C\) consists of the native temporal constraints of the initial CSTNUR plus \((\phi_1, \Box) \land \cdots \land (\phi_{13}, \Box)\) that can be compacted as \(\text{CNF}(\phi_1) \land \cdots \land (\phi_{13}), \Box)\).

After that, dynamic controllability of the CDTNU can be checked by using the methods in [14]. 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 \(- (X_{r_i} = X)\) is equivalent to \(X_{r_i} = h^*\) (like we did for clocks modeling Boolean propositions in the TGA encodings where \(- (bP < \hat{c})\) becomes \(bP = \hat{c}\)).

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

In [42], the authors 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 [24]. 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 seem 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.
In [21], a workflow and a fragment of Temporal Role-Based Access Control (TRBAC) [6] 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) were proposed in [20]. ACTNs 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.

In [45], the authors 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) [41] 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 [45] 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.

In [53], the authors 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) [23] 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 in [55, 56] with the proposal of Constraint Networks Under Conditional Uncertainty (CNCUs). 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 de-
scribed 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 [14] for CSTNUs. We proved that such mapping is correct and can be determined in polynomial time.

References


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.

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. The example is available at [http://regis.di.univr.it/FlightExample.tar.bz2](http://regis.di.univr.it/FlightExample.tar.bz2). We took advantage of Boolean variables to represent propositions and the $RA$ relation.

We used a FreeBSD virtual machine running on top of a VMware ESXi hypervisor 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 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

\[ p = \top \] (resp., \( p = \bot \)) means \( bP = \hat{c} \) (resp., \( bP < \hat{c} \)). For each \( (u, X) \in RA \), \( uX = \top \) means \( uX > \hat{c} \) if \( cX < \hat{c} \) (\( u \) executed \( X \)), \( uX = \hat{c} \) if \( cX = \hat{c} \) (\( u \) is available), whereas \( uX = \bot \) means \( uX = cX \) if \( cX < \hat{c} \) (\( u \) did not execute \( X \)) or \( uX < cX \) if \( cX = \hat{c} \) (\( u \) has been blocked).

\[ 7 \] For each proposition \( p \), \( p = \top \) (resp., \( p = \bot \)) means \( bP = \hat{c} \) (resp., \( bP < \hat{c} \)). For each \( (u, X) \in RA \), \( uX = \top \) means \( uX > \hat{c} \) if \( cX < \hat{c} \) (\( u \) executed \( X \)), \( uX = \hat{c} \) if \( cX = \hat{c} \) (\( u \) is available), whereas \( uX = \bot \) means \( uX = cX \) if \( cX < \hat{c} \) (\( u \) did not execute \( X \)) or \( uX < cX \) if \( cX = \hat{c} \) (\( u \) has been blocked).

\[ 8 \] Plenty, since UPPAAL-TIGA is compiled for 32bit architectures.
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.
Figure 14: Modeling and validation of Figure 6 with UPPAAL-TIGA.