Sound and Complete Algorithms for Checking the Dynamic Controllability of Temporal Networks with Uncertainty, Disjunction and Observation

Alessandro Cimatti*, Luke Hunsberger†, Andrea Micheli‡, Roberto Posenato§ and Marco Roveri¶
*Fondazione Bruno Kessler – Trento, Italy – Email: cimatti@fbk.eu
†Vassar College – Poughkeepsie, NY USA – Email: hunsberg@cs.vassar.edu
‡Fondazione Bruno Kessler & University of Trento – Trento, Italy – Email: amicheli@fbk.eu
§University of Verona – Verona, Italy – Email: roberto.posenato@univr.it
¶Fondazione Bruno Kessler – Trento, Italy – Email: roveri@fbk.eu

***Abstract*—Temporal networks are data structures for representing and reasoning about temporal constraints on activities. Many kinds of temporal networks have been defined in the literature, differing in their expressiveness. The simplest kinds of networks have polynomial algorithms for determining their consistency or controllability, but corresponding algorithms for more expressive networks (e.g., those that include observation nodes or disjunctive constraints) have so far been unavailable. However, recent work has introduced a new approach to such algorithms based on translating temporal networks into Timed Game Automata (TGAs) and then using off-the-shelf software to synthesize execution strategies—or determine that none exist. So far, that approach has only been used on Simple Temporal Networks with Uncertainty, for which polynomial algorithms already exist. This paper extends the temporal-network-to-TGA approach to accommodate observation nodes and disjunctive constraints. Insodoing the paper presents, for the first time, sound and complete algorithms for checking the dynamic controllability of these more expressive networks. The translations also highlight the theoretical relationships between various kinds of temporal networks and the TGA model. The new algorithms have immediate applications in the workflow models being developed to automate business processes, including in the health-care domain.**

I. Introduction

Constraint-based temporal reasoning has been widely used in many different applications across many different domains. Over the years, different formalisms have been presented to address specific requirements that frequently arise in real-world applications. The most commonly used formalism is probably the Simple Temporal Network (STN), in which a set of real-valued variables, called time-points, are subject to binary difference constraints [1]. Recently, a significant amount of research has focused on temporal reasoning in the presence of uncertainty. Temporal uncertainty arises, for example, in AI planning when the durations of some activities (i.e., the durations of some temporal intervals) are not controlled by the plan executor (or agent), but instead are only observed in real time as the activities complete. In such settings, the executor seeks a dynamic *strategy* for executing the controllable time-points such that all relevant constraints will necessarily be satisfied no matter how the uncertain durations turn out. To accommodate this kind of uncertainty, STNs have been augmented to include *contingent links,* where each contingent link represents an interval whose duration is bounded but uncontrollable; the resulting network is called a *Simple Temporal Network with Uncertainty* (STNU) [2]. The most important property of an STNU is whether it is *dynamically controllable* (DC)—that is, whether there exists a strategy for executing the controllable time-points, that depends only on knowing the outcomes of uncontrollable events up to the present time, such that all relevant constraints are guaranteed to be satisfied no matter how the durations of the contingent links turn out. Polynomial algorithms for checking the dynamic controllability property [3], [4], [5], [6] and run-time algorithms for generating an execution strategy in real-time [7], [8] have been presented in the literature.[1]

Although STNUs have been successful in some domains, many domains require a richer set of constraints. For example, in the health-care domain, where workflow management systems are being developed to automate medical-treatment processes, medical tests for any given patient frequently generate information in real time that can affect which pathway that patient will follow [9]. The system must guarantee that any possible execution of the workflow strictly satisfies all specified temporal constraints no matter which test outcomes are observed. A Conditional Simple Temporal Network with Uncertainty (CSTNU) has been introduced to represent the temporal features of workflows, and the dynamic controllability property—which captures the temporal *safety* of workflows—has been defined for CSTNUs [10]. Although some progress has been made toward a DC-checking algorithm for arbitrary CSTNUs [11], a *sound-and-complete* DC-checking algorithm for CSTNUs has not yet been found.

Disjunctive constraints can also arise in workflow management systems, for example, when two tests cannot be done simultaneously, but can be done in either order. Strong, dynamic and weak controllability have been defined for Disjunctive Temporal Networks with Uncertainty (DTNUs) [12], and algorithms for checking the strong and weak controllablitity properties for DTNUs have been presented [13], [14]; however, a dynamic controllability algorithm has only been presented for a subclass of DTNUs [12].

[1]Other controllability properties—most notably *weak controllability* and *strong controllability*—have also been studied. They make different assumptions about when the durations of the contingent links become known to the executor and, thus, have fewer practical applications.

This paper capitalizes on a new approach [15] to checking the dynamic controllability of temporal networks that first translates the given network into a provably equivalent Timed Game Automaton (TGA) [16] and then uses the UPPAAL-TIGA software [17] to synthesize a viable execution strategy for that TGA—or confirm that no such strategy exists. So far, this approach has only been used on STNUs, for which polynomial-time DC-checking algorithms already exist.[2] This paper extends the approach to accommodate both the observation time-points of CSTNUs and disjunctive constraints of DTNUs. In so doing, it presents the first *sound-and-complete* DC-checking algorithm for temporal networks having contingent links, disjunctive constraints and observation time-points in any combination—an important advance that is expected to have immediate practical applications in workflow management systems. Furthermore, the translation of such networks into TGAs highlights important theoretical relationships between the different kinds of temporal reasoning frameworks.

Paper structure. Section II presents relevant background on temporal networks and the recent work on translating temporal networks into Timed Game Automata. Section III presents the main contribution of this paper: our encoding of more expressive temporal networks—CSTNUs and DTNUs—into TGAs to generate, for the first time, sound-and-complete dynamic controllability checking algorithms for these kinds of temporal networks. Finally, Section IV draws some conclusions and discusses future work.

II. Background

This section first describes a model of workflows that is being developed to automate medical-treatment processes in the healthcare domain. Next, it discusses relevant background information on Simple Temporal Networks (STNs), Simple Temporal Networks with Uncertainty (STNUs), and Conditional Simple Temporal Networks with Uncertainty (CSTNUs), the latter having been presented in the literature as the theoretic and algorithmic foundation for the temporal aspects of a reasonably expressive class of workflows. In the cases of STNs and STNUs, it illustrates the recently reported technique of translating temporal networks into Timed Game Automata (TGAs). The rest of the paper then addresses our main contribution—namely, extending this technique to cover CSTNUs and networks with disjunctive constraints to generate sound-and-complete DC-checking algorithms for these more expressive networks.

A. Workflows

A workflow is an abstract model for representing, coordinating and controlling complex processes. A *workflow management system* (WfMS) is a software suite that supports the automatic execution of workflows [18]. Although workflows are being applied to a variety of businesses, the research presented in this paper has been motivated by the use of workflows to automate medical-treatment processes in the healthcare domain.

[2]Despite the exponential run-time of the UPPAAL-TIGA software, the fully-fleshed-out strategies that it generates are useful in domains (e.g., spacecraft) where the executor has extremely limited computational power. The execution strategy can be generated off-line, in advance of execution; it can then be executed quickly during run-time. In contrast, prior approaches to generating execution decisions for DC STNUs required more computation due to the need of propagating constraints and reasoning at run-time.

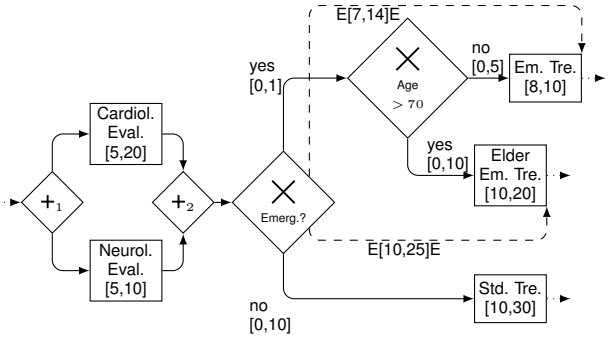


Fig. 1. An excerpt of a simplified triage workflow schema

In a workflow management system, the management of temporal aspects is critical. The literature contains many proposals for extending workflow models to represent and manage the most important kinds of temporal constraints that arise in various domains [19], [20], [21]. This paper focuses on the conceptual model proposed by Combi et al. [22], where a workflow is specified by a *workflow schema*: a directed graph where nodes represent activities, and arcs represent control flows that define activity dependencies on the execution order. Fig. 1 illustrates a small portion of such a workflow schema.

There can be two types of activities in a workflow graph: *tasks* and *connectors*. Tasks represent elementary work units to be executed by external agents (e.g., doctors); connectors represent internal activities executed by the WfMS to coordinate the execution of the tasks. In the graph, each task is represented by a box containing a name (e.g., Neurol. Eval.) and a range (e.g., [5, 10]) that constrains the task's duration during execution. Each connector is represented by a diamond that, similarly, may contain a temporal range constraining its duration. Additional information associated with a connector depends on its type—*split* or *join*—as discussed below.

The arcs in a workflow graph are labeled directed edges that impose ordering constraints among nodes and, optionally, temporal delays. For example, an arc from a predecessor node N_1, to a successor node N_2, with the label [5,10], specifies that N_2 must start between 5 and 10 time units after the completion of N_1. A *split* connector has one incoming arc and multiple outgoing arcs. After the execution of the split connector node, one or more successor nodes must be considered for execution, depending on whether the split connector type is *parallel, alternative* or *conditional. Join* connectors are the mirror image of *split* connectors, having multiple incoming arcs but only one outgoing arc. Join connectors effectively close branching paths opened by split connectors.

The workflow in Fig. 1 represents a simplified *triage* process for a hospital emergency room. In the figure, all temporal ranges are in minutes; parallel connectors are identified by + and conditional connectors by ×; and each connector is presumed to have a temporal range of [0,0]. An incoming patient is first evaluated from the cardiological and neurological points of view: two parallel tasks that can be done in either order, but cannot overlap. The subsequent treatment path depends on the observation of two boolean conditions: (1) whether it is

an emergency (Emerg.?); and (2) whether the patient is old (Age $>$ 70?). In the non-emergency case, the patient is given a standard treatment (Std. Tre.). But for an emergency, depending on the patient's age, the patient is given either emergency treatment (Em. Tre.) or elder emergency treatment (Elder Em. Tre.). In other words, the conditional connectors, $\times_{\text{Emerg.?}}$ and $\times_{(\text{Age}>70)}$, each split the flow into alternative pathways based on the observation of their corresponding boolean conditions.

Finally, because some activities in a workflow may be subject to important temporal constraints, workflows also include dashed edges that represent temporal constraints. Such constraints may relate the starting or ending times of the source and target nodes in any combination. For example, the dashed edge labeled by E[7,14]E specifies that the end of the Em. Tre. task must occur between 7 and 14 minutes after the end of the $\times_{\text{Emerg.?}}$ connector. The other dashed edge similarly specifies that the end of the Elder Em. Tre. task must occur between 10 and 25 minutes after the end of the $\times_{\text{Emerg.?}}$ connector.

Hunsberger et al. [10] defined a Conditional Simple Temporal Network with Uncertainty (CSTNU) to represent the most important temporal features of workflow schemata. Below, the relevant background from Simple Temporal Networks (STNs) to Simple Temporal Networks with Uncertainty (STNUs) to CSTNUs is presented.

B. Simple Temporal Networks and STNUs

Definition 1 (STN [1])**.** *A* Simple Temporal Network *(STN) is a pair $(\mathcal{T}, \mathcal{C})$ where: $\mathcal{T}$ is a set of real-valued variables called* time-points*, and $\mathcal{C}$ is a set of binary constraints, each of the form, $Y - X \leq \delta$, for some $X, Y \in \mathcal{T}$ and $\delta \in \mathbb{R}$. An STN is* consistent *if there exists a set of variable assignments to the time-points in $\mathcal{T}$ that satisfy all of the constraints in $\mathcal{C}$.*

Although STNs have been widely used in planning and scheduling, they do not capture the dynamism of domains in which the durations of some actions (i.e., some temporal intervals) are not controlled by the executor (or agent), but are only observed, in real time, as those actions complete. An STNU augments an STN to include *contingent links* that represent such uncontrollable (but bounded) intervals.

Definition 2 (STNU [23])**.** *An STNU is a tuple $(\mathcal{T}, \mathcal{C}, \mathcal{L})$ such that: (1) $(\mathcal{T}, \mathcal{C})$ is an STN; (2) $\mathcal{T}$ is partitioned into two sets, $\mathcal{T}_f$ and $\mathcal{T}_u$, of* free *and* uncontrollable *time-points; and (3) $\mathcal{L}$ is a set of* contingent links*, each of the form, (A, ℓ, u, C), where $A \in \mathcal{T}, C \in \mathcal{T}_u$, and $0 < \ell < u < \infty$.*

Often the notation $Y - X \in [a, b]$ is used as an abbreviation for $Y - X \leq b$ and $X - Y \leq -a$. Each contingent link, (A, ℓ, u, C), represents a temporal interval from A to C whose duration is uncontrollable, but bounded by $C - A \in [\ell, u]$. A is called the *activation* time-point, while C is called the *contingent* time-point.

The most important property of an STNU is whether it is *dynamically controllable* (DC), that is whether there exists a strategy for executing the free time-points in $\mathcal{T}_f$ such that all constraints in $\mathcal{C}$ will necessarily be satisfied no matter how the durations of the contingent links in $\mathcal{L}$ turn out [23]. Note that the execution strategy is *dynamic* in that it can react to observations of contingent time-points, but only after some positive delay. Polynomial algorithms for checking the DC property have been presented in the literature [3], [4], [5].

C. Conditional Simple Temporal Networks with Uncertainty

A CSTNU augments an STNU to include propositional letters that represent boolean conditions whose truth values are observed in real time, during execution.[3] *Labels* comprising conjunctions of (positive or negative) propositional letters can be associated with time-points and temporal constraints. A time-point with propositional label ℓ is only executed in scenarios where ℓ is true; and a constraint labeled by ℓ only applies in scenarios where ℓ is true. Some of the time-points in a CSTNU are called *observation* time-points. Each observation time-point X has an associated propositional letter, whose truth value is obtained in real time when X is executed.

Definition 3 (Labels [24])**.** *Given a set P of propositional letters, a* label *is any (possibly empty) conjunction of (positive or negative) literals from P. The* label universe *of P, denoted by P^*, is the set of all labels with literals drawn from P.*

Definition 4 (CSTNU [10])**.** *A* Conditional Simple Temporal Network with Uncertainty *(CSTNU) is a tuple, $(\mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P, \mathcal{L})$, where:*

(1) P is a set of propositional letters;
(2) $\mathcal{T}$ is a set of time-points;
(3) $\mathcal{OT} \subseteq \mathcal{T}$ is a set of observation time-points;
(4) $\mathcal{O} : P \to \mathcal{OT}$ is a bijection that assigns propositional letters to observation time-points;
(5) $L : \mathcal{T} \to P^*$ is a function assigning labels to time-points;
(6) $\mathcal{C}$ is a set of labeled *temporal constraints of the form, $\langle Y - X \leq \delta,\ \ell \rangle$, where $X, Y \in \mathcal{T}$ and $\ell \in P^*$; and*
(7) ignoring any labels, $(\mathcal{T}, \mathcal{C}, \mathcal{L})$ is an STNU.[4]

For the subclass of workflow schemata having no disjunction constraints (like the two parallel non-overlapping tasks of Fig. 1), Hunsberger et al. [10] presented a method of encoding the temporal information from each workflow schema into a CSTNU with the aim of rigorously analyzing and validating the temporal safety of the schemata. In particular, the CSTNU for a given workflow schema is obtained as follows. First, each task is represented by a contingent link. Second, each connector is represented by a pair of (starting and ending) time-points, linked by a duration constraint. Third, each arc is represented by a duration constraint. Fourth, the ending time-point for each conditional split connector is represented by an observation time-point for a proposition whose possible values correspond to the different branching decisions of that connector.[5] Fifth, the propositional label for each time-point is obtained by accumulating the propositional literals along the relevant pathway. Sixth, the propositional label for each temporal constraint is obtained by conjoining the labels on the associated time-points. Finally, if the duration range of a connector is $[0, 0]$, then the connector can be represented by a single time-point.

[3]The "C" in CSTNU derives from the "C" in the *Conditional Temporal Problem* (CTP) introduced by Tsamardinos et al. [24]. A CSTNU extends both Conditional Temporal Networks (CTNs) and STNUs.

[4]There are some additional technical conditions on CSTNUs that are omitted for expository convenience [10].

[5]Without loss of generality, this paper considers only binary branching.

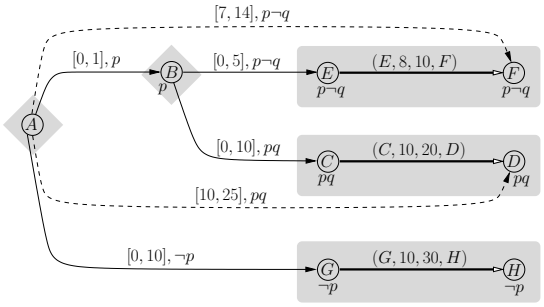


Fig. 2. The CSTNU obtained from a portion of the workflow from Fig. 1

Fig. 2 shows the CSTNU obtained in this way for the portion of the workflow from Fig. 1 that excludes the disjunctive Cardiol. Eval. and Neurol. Eval. tasks. To facilitate comparison with the original workflow, the contingent links derived from the three workflow tasks have been shaded with oblong boxes, the observation time-points derived from the split connectors have been shaded with diamond-shaped boxes, and the edges derived from temporal constraints are dashed. The observation time-point A yields the proprosition p (i.e., Emerg?); and B yields the proposition q (i.e., Age $>$ 70?). The contingent link $(E, 8, 10, F)$ corresponds to the Em. Tre. task; $(C, 10, 20, D)$ corresponds to Elder Em. Tre.; and $(G, 10, 30, H)$ corresponds to Std. Tre.

Hunsberger et al. [10] also provided a definition of the critical property of dynamic controllability for CSTNUs, generalizing the dynamic controllability of STNUs [23] and the *dynamic consistency* of a CTP [24]. In brief, a CSTNU is dynamically controllable if there exists a strategy for executing the free time-points in the network such that all constraints in $\mathcal{C}$ are guaranteed to be satisfied no matter how the durations of the contingent links turn out, and no matter how the observations of the various propositions turn out, in real time—with the caveat that in any given scenario, only the time-points whose labels are true in that scenario need to be executed, and only the constraints whose labels are true in that scenario need to be satisfied. Subsequent work yielded a variety of constraint propagation rules for CSTNUs, which led to a sound-*but-not-complete* DC-checking algorithm for CSTNUs [11]. Since the temporal safety of a workflow schema corresponds directly to the dynamic controllability of the underlying CSTNU, providing a sound-and-complete DC-checking algorithm for CSTNUs remained an important open problem.

D. DTNUs & Dynamic Controllability

Another important way of extending STNUs is to include disjunctive constraints. The result is a *Disjunctive Temporal Network with Uncertainty* (DTNU) [25]. Extending STNUs in this way is analogous to extending the Simple Temporal Problem (STP) to the Disjunctive Temporal Problem (DTP) [26]. Disjunctions frequently arise in practice: workflows in the healthcare domain frequently involve activities whose executions cannot overlap due to conflicting requirements. For example, the Cardiol. Eval. and Neurol. Eval. tasks from the workflow in Fig. 1 have uncontrollable durations bounded by $[5, 20]$ and $[5, 10]$, respectively. These tasks can be done in any order, but cannot overlap. To retain maximal flexibility, we want to constrain these tasks to be non-overlapping without imposing any a priori order on them.

Definition 5. *A DTNU is a triple, $(\mathcal{T}, \mathcal{C}, \mathcal{L})$, where: $\mathcal{T}$ is a set of real-valued variables called* time-points, *partitioned into the sets, $\mathcal{T}_f$ and $\mathcal{T}_u$, of* free *and* uncontrollable *time-points; $\mathcal{C}$ is a set of constraints, each obtained as the arbitrary boolean combination of atoms in the form, $Y - X \leq \delta$, for some $X, Y \in \mathcal{T}$ and $\delta \in \mathbb{R}$; and $\mathcal{L}$ is a set of* contingent links*, each of the form, $(A, \mathcal{B}, C)$, where $A \in \mathcal{T}$, $C \in \mathcal{T}_u$, and $\mathcal{B}$ is a finite set of pairs (ℓ, u) such that $0 < \ell < u < \infty$; and for each (ℓ_i, u_i) and (ℓ_j, u_j), either $(\ell_i > u_j)$ or $(u_i < \ell_j)$.*

Analogously to the STNU case, a contingent link, $(A, \{(\ell_1, u_1), \cdots, (\ell_n, u_n)\}, C)$, represents a temporal interval from A to C whose duration is uncontrollable, but constrained to lie within a union of disjoint intervals. That is, $C - A \in [\ell_1, u_1] \cup \cdots \cup [\ell_n, u_n]$. This is useful to model periodic activities whose windows of opportunity have certain degrees of uncertainty. An STNU is the particular case of DTNU in which all the $\mathcal{B}$ sets are singletons and conjunction is the only allowed boolean operator in the constraints belonging to $\mathcal{C}$. The dynamic controllability problem for DTNUs is also defined analogously to the STNU case [13], [12]: the aim is to find a strategy for executing the free time-points that will guarantee the satisfaction of all constraints in $\mathcal{C}$ no matter how the contingent durations turn out—within their specified disjunctive bounds.

The example of the parallel Cardiol. Eval. and Neurol. Eval. tasks can be encoded as a DTNU $(\mathcal{T}_f \cup \mathcal{T}_u, \mathcal{C}, \mathcal{L})$ as follows. First, let C_s, C_e, N_s and N_e be the starting and ending times for these two tasks; then, $\mathcal{T}_f \doteq \{C_s, N_s\}$ and $\mathcal{T}_u \doteq \{C_e, N_e\}$. The constraint set $\mathcal{C}$ contains just one disjunctive constraint:

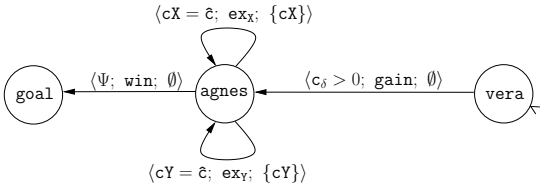
$$\mathcal{C} \doteq \{(N_e - C_s \leq 0) \vee (C_e - N_s \leq 0)\}.$$

Finally the set of contingent links is given by:

$$\mathcal{L} \doteq \{(C_s, \{(5, 20)\}, C_e), (N_s, \{(5, 10)\}, N_e)\}.$$

E. Translating Temporal Networks into Timed Game Automata

Recently, Cimatti et al. [15] presented a new approach for synthesizing execution strategies for STNUs. The approach is based on a novel translation of STNUs into Timed Game Automata (TGAs) that preserves the execution semantics. Given any STNU $\mathcal{S}$, the following steps are taken: (1) translate $\mathcal{S}$ into an equivalent TGA, $\Theta_{\mathcal{S}}$; (2) use the UPPAAL-TIGA software [17], [27] to synthesize a *dynamic* execution strategy for $\Theta_{\mathcal{S}}$; and (3) convert that strategy into an executable format that requires minimal computational overhead during runtime. Although this approach can take exponential time—as compared to the polynomial time required by existing DC-checking algorithms for STNUs—it clarifies the deep theoretical relationships between STNUs and TGAs, thereby paving the way for extensions of the approach to more expressive kinds of temporal networks (e.g., CSTNUs and DTNUs) for which no sound-and-complete DC-checking algorithms have yet been discovered. Furthermore, the third step in the process will provide similar benefits in the extensions of this approach when used in domains where the executor has limited computational resources. In such cases, it is important to distinguish the exponential time required to generate the execution strategy



Note: $\Psi = (\mathtt{cX} < \hat{\mathtt{c}}) \wedge (\mathtt{cY} < \hat{\mathtt{c}}) \wedge (\mathtt{cX} - \mathtt{cY} \leq 5)$

Fig. 3. The translation of the sample STN into a Timed Automaton

in advance from the minimal time required during execution to apply the strategy.

For expository purposes, the simpler translation from an STN to a Timed Automaton (TA) is described first.

Definition 6 (Timed Automaton [28])**.** *A* Timed Automaton *(TA) is an automaton that includes the following features:*

- *L, a finite set of locations (or states), one of which is called the* initial location;
- *$\mathcal{X}$, a finite set of real-valued* clocks; *and*
- *E, a finite set of* labeled transitions.

During any run *of the TA, all clocks start out at zero and advance uniformly. The clock values affect the operation of the TA as follows. First, any transition has the form, $(\ell_i, G, N, R, \ell_j)$, where ℓ_i is the starting location, ℓ_j is the ending location, G is a* guard *(i.e., a conjunction of atomic constraints that restrict when the transition can be taken from ℓ_i to ℓ_j), N is a* name *for the transition (often called an* action*), and R is a subset of clocks that are* reset *to 0 whenever the transition is taken.*[6] *Second, each location, ℓ, may have an associated* invariant *(i.e., a conjunction of atomic constraints that restricts how long the TA can stay in that location). The atomic constraints in a TA each have the form, $Y - X \bowtie k$, where X and Y are clocks, k is an integer, and $\bowtie$ is one of: $<, >, \leq, \geq$ or $=$.*

Consider the STN, $\mathcal{S} = (\{X, Y\}, \{Y - X \leq 5\})$. It has two time-points, X and Y, and one temporal constraint, $Y - X \leq 5$. As illustrated in Fig. 3, this STN can be translated into a Timed Automaton having the following features:

$$\begin{array}{ll} \text{Locations:} & L = \{\mathtt{vera}, \mathtt{agnes}, \mathtt{goal}\} \\ \text{Clocks:} & \mathcal{X} = \{\mathtt{cX}, \mathtt{cY}, \hat{\mathtt{c}}, \mathtt{c}_\delta\} \\ \text{Transitions:} & E = \{\mathtt{gain}, \mathtt{ex_X}, \mathtt{ex_Y}, \mathtt{win}\} \end{array}$$

There are four clocks in this TA. $\hat{\mathtt{c}}$ starts out at 0 and is never reset; it is used to keep track of global time. cX and cY are reset exactly once each: the instants that X and Y are executed, respectively. And $\mathtt{c}_\delta$ is used simply to ensure that the transition into agnes can only be taken after some positive delay.

The location agnes represents a state in which Agnes (an agent) can execute the time-points X and Y. The transition $\mathtt{ex_X}$ has the label, $\langle \mathtt{cX} = \hat{\mathtt{c}};\ \mathtt{ex_X};\ \mathtt{cX} \rangle$. Its guard stipulates that $\mathtt{cX} = \hat{\mathtt{c}}$, but taking this transition causes cX to be reset. Therefore, this transition can be taken at most once. Furthermore, once X is executed, its execution time is forever given by the fixed difference, $\hat{\mathtt{c}} - \mathtt{cX}$. (These clocks can never be reset at any future time.) Analogous remarks apply to the transition $\mathtt{ex_Y}$. The transition win has the guard, $\Psi = (\mathtt{cX} < \hat{\mathtt{c}}) \wedge (\mathtt{cY} < \hat{\mathtt{c}}) \wedge (\mathtt{cX} - \mathtt{cY} \leq 5)$. The constraints, $(\mathtt{cX} < \hat{\mathtt{c}})$ and $(\mathtt{cY} < \hat{\mathtt{c}})$, stipulate that the time-points, X and Y, must have been executed. The constraint, $(\mathtt{cX} - \mathtt{cY} \leq 5)$, stipulates that the execution times for X and Y must satisfy $(Y - X \leq 5)$.[7] Thus, the win transition can only be taken in cases where the STN has been successfully executed.

[6] In the graphs for TAs illustrated in this paper, each transition, $(\ell_i, G, N, R, \ell_j)$, is represented by a directed edge from ℓ_i to ℓ_j, labeled by $\langle G; N; R \rangle$. Thus, the labels on TA transitions are quite different from the propositional labels that appear in CSTNU graphs.

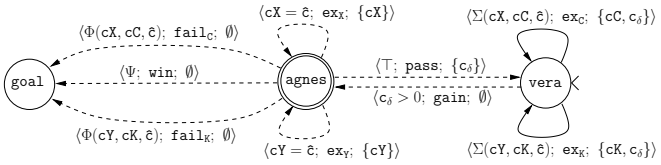


Fig. 4. The translation of the sample STNU into a TGA

The goal for Agnes is to get from the initial state, vera, to the goal state, goal. Given the guard, Ψ, on the win transition, the only way that Agnes can do this is to first execute X and Y in a way that satisfies the constraint, $Y - X \leq 5$, and then take the win transition to the goal state.

Any STN can be translated into a Timed Automaton in this way. No matter how many time-points or constraints it has, the corresponding TA will have the same three locations, the same transition from vera to agnes, and the analogous win transition from agnes to goal. The TA will have the global clock $\hat{\mathtt{c}}$, and the delay clock $\mathtt{c}_\delta$. For each time-point, X, there will be one clock, cX, and one self-loop at agnes. Finally, the guard on the win transition will be given by:

$$\Psi = (\wedge_i (\mathtt{cX}_i < \hat{\mathtt{c}})) \wedge (\wedge_{(X_j - X_i \leq \delta_{ij})} (\mathtt{cX}_i - \mathtt{cX}_j \leq \delta_{ij})).$$

Definition 7 (Timed Game Automaton [16])**.** *A* Timed Game Automaton *(TGA) is a Timed Automaton whose transitions are partitioned into* controllable *and* uncontrollable *transitions.*

A TGA can be used to model a two-player game between Agnes (an agent) and Vera (the environment) that is governed by various temporal constraints. Any STNU can be translated into a TGA [15]. The translation is illustrated below using the STNU $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$, where:

$$\begin{array}{l} \mathcal{T} = \{X, Y, C, K\}; \\ \mathcal{C} = \{Y - X \leq 5,\ C - K \leq 10\}; \text{ and} \\ \mathcal{L} = \{(X, 3, 9, C),\ (Y, 4, 7, K)\}. \end{array}$$

This STNU has four time-points, two constraints, and two contingent links. Note that X and Y are *free* time-points, controlled by the agent, while C and K are *contingent* time-points controlled by the environment. As shown in Fig. 4, this STNU can be translated into a TGA by making the following incremental changes to the automaton from Fig. 3.

- For each contingent time-point C, add a clock cC, and a loop at vera representing the action of Vera executing C.
- Make agnes an *urgent* location—that is, Agnes cannot wait in that location for any positive time: she must exe-

[7] Note that $Y - X = (\hat{\mathtt{c}} - \mathtt{cY}) - (\hat{\mathtt{c}} - \mathtt{cX}) = \mathtt{cX} - \mathtt{cY}$.

cute whatever time-points she wants to execute, and then instantaneously take a transition to some other location.[8]

- For each contingent link, (A, u, v, C), add a transition from `agnes` to `goal` that is only enabled if Vera fails to execute C such that $C - A \in [u, v]$.
- Give Agnes control over all transitions except for the loops at `vera`. In the figure, the transitions controlled by Agnes are shown as dashed.[9]

A run of this TGA begins at the initial location `vera`. Whenever Agnes wants to execute one or more time-points, she takes the `gain` transition to the `agnes` location, executes her desired time-points, and then immediately returns to `vera`. Thus, the TGA remains at `vera`, except for instantaneous transitions to `agnes` whenever Agnes wants to execute a time-point. As a result, Vera can execute her contingent time-points whenever she wants—as long as the appropriate guards are satisfied. The guards specify that the contingent durations must lie within their corresponding bounds. For example, the guard on the $\texttt{ex}_\texttt{C}$ transition is:

$$\Sigma(\texttt{cX}, \texttt{cC}, \hat{\texttt{c}}) = (\texttt{cX} < \hat{\texttt{c}}) \wedge (\texttt{cC} = \hat{\texttt{c}}) \wedge (\texttt{cX} \geq 3) \wedge (\texttt{cX} \leq 9)$$

where 3 and 9 are the lower and upper bounds on the contingent link $(X, 3, 9, C)$. The clock $\texttt{c}_\delta$ ensures that Agnes can react only to the execution of contingent time-points after some positive delay—a key feature of the execution semantics for STNUs. Finally, the guard on the `win` transition is:

$$\begin{aligned}\Psi = (\texttt{cX} < \hat{\texttt{c}}) \wedge (\texttt{cY} < \hat{\texttt{c}}) \wedge (\texttt{cC} < \hat{\texttt{c}}) \wedge (\texttt{cK} < \hat{\texttt{c}}) \\ \wedge\ (\texttt{cX} - \texttt{cY} \leq 5) \wedge (\texttt{cK} - \texttt{cC} \leq 10)\end{aligned}$$

representing that all time-points have been executed, and all constraints have been satisfied.

This translation has been proven to correctly capture the execution semantics of STNUs. The TGA translation has a winning strategy for Agnes if and only if the original STNU is dynamically controllable [15]. Although the DC-checking algorithm generated in this way takes exponential time in the worst case, while there already is a polynomial-time DC-checking algorithm for STNUs, this new approach has the following benefits:

- ⋆ it explicates the relationships between STNUs and TGAs, especially with regard to instantaneous reactivity;
- ⋆ the strategies it synthesizes require minimal computation during the subsequent execution of the network, which is critical in some domains (e.g., spaceships); and
- ⋆ (most important for this paper) the translation technique provides a foundation for creating sound-and-complete DC-checking algorithms for more expressive networks.

III. DC-checking Algorithms for More Expressive Temporal Networks

This section presents our approach for translating more expressive temporal networks into TGAs with the aim of producing sound-and-complete dynamic controllability algorithms for those networks. (Such algorithms have never been presented before.) Starting with an STNU and its translation into a TGA, this section incrementally adds two new features: (1) observation nodes from CSTNUs; and (2) disjunctive constraints from DTNUs. In each case, the TGA-translation is extended to accommodate the new feature, while preserving the desired execution semantics. The desired DC-checking algorithm for any network having contingent links, observation nodes and disjunctive constraints, in any combination, is generated by first translating the network into the corresponding TGA, and then using the UPPAAL-TIGA software [17], [27] to synthesize a valid execution strategy—or confirm that no such strategy exists. Since the TGA translation preserves the execution semantics, and the algorithm implemented in UPPAAL-TIGA is sound and complete, the DC-checking algorithm constructed in this way is also sound and complete.

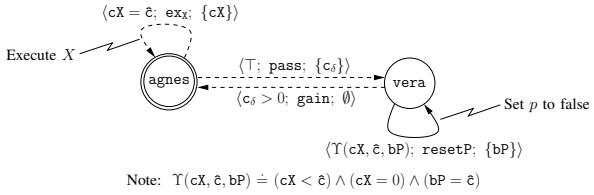


Fig. 5. An observation node X whose associated proposition is p

A. Observation Nodes

As described in Sec. II-C, an observation node is a time-point whose execution generates a truth value for an associated proposition (equivalently, a boolean variable). For example, let X be an observation time-point whose execution establishes the truth value of the proposition p. Note that before X is executed (i.e., while $\texttt{cX} = \hat{\texttt{c}}$), the truth value of p is unknown, but after p is executed, its truth value must be either true or false. This feature can be accommodated in a TGA by introducing a new clock `bP` (which stands for "boolean p"), whose value is meaningful only after X has been executed. After X has executed, if $\texttt{bP} = \hat{\texttt{c}}$, then p shall be interpreted as being true; but if $\texttt{bP} < \hat{\texttt{c}}$ (i.e., if `bP` has been reset), then p shall be interpreted as being false. As shown in Fig. 5, at the instant X is executed by Agnes, the guard on the `resetP` transition (a loop at `vera`) gives Vera precisely one opportunity to reset `bP` (i.e., to make p false). The guard on `resetP` is:

$$\Upsilon(\texttt{cX}, \hat{\texttt{c}}, \texttt{bP}) \doteq (\texttt{cX} < \hat{\texttt{c}}) \wedge (\texttt{cX} = 0) \wedge (\texttt{bP} = \hat{\texttt{c}})$$

which represents that X has been executed *now,* but `bP` has not yet been reset. If Vera does not take this opportunity, then `bP` shall forever be equal to $\hat{\texttt{c}}$ (i.e., p shall forever be true).

Next, the `win` transition that represents the "all time-points executed and all constraints satisfied" condition is changed so that it emanates not from the `agnes` location, but from the `vera` location—and with a guard that includes the constraint $\texttt{c}_\delta > 0$. This is done to ensure that Vera always gets her opportunity to set the truth value of any proposition corresponding to a just executed observation time-point. (Otherwise, Agnes might surreptitiously execute an observation time-point and then take the `win` transition before Vera has a chance to set the truth value of the corresponding proposition.)

[8] Urgent locations are commonly used with TGAs. An urgent location, L, can be implemented using a dedicated clock, c, that is reset to zero upon entering L, and by adding the guard, $c \leq 0$, to each transition leaving L.

[9] Agnes is given control over the *uncontrollable* transitions, while Vera is given the *controllable* transitions. This swapping of the traditional roles is necessary to properly capture the execution semantics for STNUs [15].

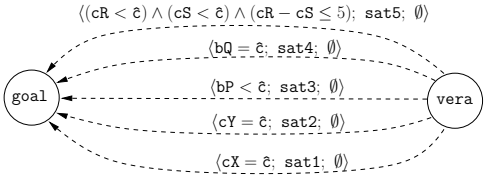


Fig. 6. The transitions capturing the sample "all time-points executed and all constraints satisfied" constraint in the scenario $p\neg q$, discussed in the text

Finally, since the nodes and constraints in a CSTNU can be labeled by conjunctions of (positive or negative) propositional letters, the "all time-points executed and all constraints satisfied" condition must be represented in a new way. To see this, consider the following example. Suppose that the time-points, R and S, and the constraint, $S - R \leq 5$, are labeled by $p\neg q$ (i.e., $p \wedge (\neg q)$). Suppose further that X and Y are the observation time-points for p and q, respectively. Then, in any scenario in which X and Y are both executed, and p is true, and q is false, success requires that both R and S be executed, and that $S - R \leq 5$ be satisfied. This can be represented by the following *conditional* constraint:

$$((\mathtt{cX} < \hat{\mathtt{c}}) \wedge (\mathtt{cY} < \hat{\mathtt{c}}) \wedge (\mathtt{bP} = \hat{\mathtt{c}}) \wedge (\mathtt{bQ} < \hat{\mathtt{c}})) \\ \Rightarrow ((\mathtt{cR} < \hat{\mathtt{c}}) \wedge (\mathtt{cS} < \hat{\mathtt{c}}) \wedge (\mathtt{cR} - \mathtt{cS} \leq 5))$$

which is equivalent to:

$$(\neg(\mathtt{cX} < \hat{\mathtt{c}})) \vee (\neg(\mathtt{cY} < \hat{\mathtt{c}})) \vee (\neg(\mathtt{bP} = \hat{\mathtt{c}})) \vee (\neg(\mathtt{bQ} < \hat{\mathtt{c}})) \\ \vee((\mathtt{cR} < \hat{\mathtt{c}}) \wedge (\mathtt{cS} < \hat{\mathtt{c}}) \wedge (\mathtt{cR} - \mathtt{cS} \leq 5))$$

Given that no clock's value can ever exceed that of the global clock $\hat{\mathtt{c}}$, this further simplifies to:

$$(\mathtt{cX} = \hat{\mathtt{c}}) \vee (\mathtt{cY} = \hat{\mathtt{c}}) \vee (\mathtt{bP} < \hat{\mathtt{c}}) \vee (\mathtt{bQ} = \hat{\mathtt{c}}) \\ \vee ((\mathtt{cR} < \hat{\mathtt{c}}) \wedge (\mathtt{cS} < \hat{\mathtt{c}}) \wedge (\mathtt{cR} - \mathtt{cS} \leq 5))$$

Since the guards on TGA transitions do not allow disjunction, the above condition can be represented using five separate transitions, one for each disjunct, as illustrated in Fig. 6.

In general, suppose that $\ell_1, \ell_2, \ldots, \ell_M$ are the M distinct labels that appear in some CSTNU. (Typically, M is much smaller than the number of labels in the label universe, P^*.) For each i, let τ_i be the set of time-points labeled by ℓ_i, and $\mathcal{C}_i$ the set of constraints labeled by ℓ_i. In addition, for each i, let θ_i be the conditional constraint that can be glossed as: "In scenarios where ℓ_i is true, all of the time-points in τ_i must be executed, and all of the constraints in $\mathcal{C}_i$ must be satisfied," as discussed in the preceding example. The desired "all time-points executed and all constraints satisfied" condition is then the conjunction: $\wedge_{i=1}^{M}\ \theta_i$. This conjunction of conditional constraints can be effectively accommodated in the TGA using a sequence of $(M+1)$ locations starting from $L_0 = \mathtt{vera}$, and ending at $L_M = \mathtt{goal}$, as follows:

$$(\mathtt{vera} = L_0) \ \rightsquigarrow\ L_1 \ \rightsquigarrow\ L_2 \ \rightsquigarrow\ \ldots \ \rightsquigarrow\ (L_M = \mathtt{goal})$$

For each i, there is a set of transitions from L_{i-1} to L_i that together represent the conditional constraint θ_i, as illustrated in Fig. 7. If, in some scenario, Agnes can follow a path through this network of transitions from vera to goal, then all of the relevant time-points must have been executed, and all of the relevant constraints must have been satisfied.

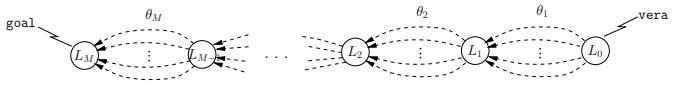


Fig. 7. Using a sequence of locations in a TGA to accommodate the "all time-points executed and all constraints satisfied" condition for a CSTNU

Fig. 8 illustrates the TGA that is obtained in this way from the sample CSTNU seen earlier in Fig. 2. In the figure, the sequence of locations, L_i, have been renamed to show the associated labels, as follows:

$$\mathtt{vera} \rightsquigarrow L_\emptyset \rightsquigarrow L_p \rightsquigarrow L_{p\neg q} \rightsquigarrow L_{pq} \rightsquigarrow (L_{\neg p} = \mathtt{goal})$$

In addition, to reduce the clutter, only the guards are shown on the transitions in this sequence.

Given that the UPPAAL-TIGA software [17], [27] extends the syntax of TGAs to include boolean and integer-valued variables that can appear in guards and be reset by transitions, an alternative translation from CSTNUs to TGAs is possible. In this alternative translation, the following variables are introduced: (1) for each time-point X, a boolean variable xX, whose value is true if and only if X has been executed; and (2) for each proposition p, an integer-valued variable p whose value is 0 prior to p being observed, -1 if p is observed to be false, and 1 if p is observed to be true. Fig. 9 illustrates the TGA that results from this alternative technique. That TGA also takes advantage of the branching structure of the original workflow to simplify the transitions from vera to goal. Instead of a long sequence of effectively disjunctive transitions, this TGA has a much simpler branching structure that mirrors the structure of the workflow. Initial runs of the UPPAAL-TIGA software on such TGAs suggests that this alternative translation of CSTNUs into TGAs is much more efficient in practice, although equivalent in theory.

B. Disjunctive Constraints

A DTNU generalizes an STNU in two different dimensions. First, the durations of contingent links can be constrained to lie within a union of disjoint intervals. Second, the free constraints can comprise arbitrary boolean combinations of difference constraints. This section shows how a DTNU $(\mathcal{T}, \mathcal{C}, \mathcal{L})$ can be translated into an equivalent TGA by making two modifications to the STNU-to-TGA translation. First, if the duration for a contingent link is constrained to lie within one of n disjoint intervals, then there will be n corresponding loops at the vera location, where the guard for each loop effectively specifies one of the allowed intervals for that contingent duration. Second, the "all time-points executed and all constraints satisfied" transition to the goal location is represented by alternative pathways through a sequence of locations from vera to goal, using a technique that generalizes that shown for CSTNUs in the preceding section.

To begin, as for an STNU, each free time-point X will have a corresponding transition, $(\mathtt{agnes}, \mathtt{cX} = \hat{\mathtt{c}}, s_X, \{\mathtt{cX}\}, \mathtt{agnes})$, that represents the execution of X by Agnes. However, for each disjunctive contingent link, $(A, \mathcal{B}, C)$, where $\mathcal{B} \doteq \{(\ell_1, u_1), \cdots, (\ell_n, u_n)\}$, there are n loop transitions: $(\mathtt{vera}, \Sigma(\mathtt{cC}, \mathtt{cA}, \hat{\mathtt{c}}, l_i, u_i), s_C, \{\mathtt{cC}, \mathtt{c}_\delta\}, \mathtt{vera})$, for $i \in [1, n]$. They represent the possible executions of C by Vera. The respective guards,

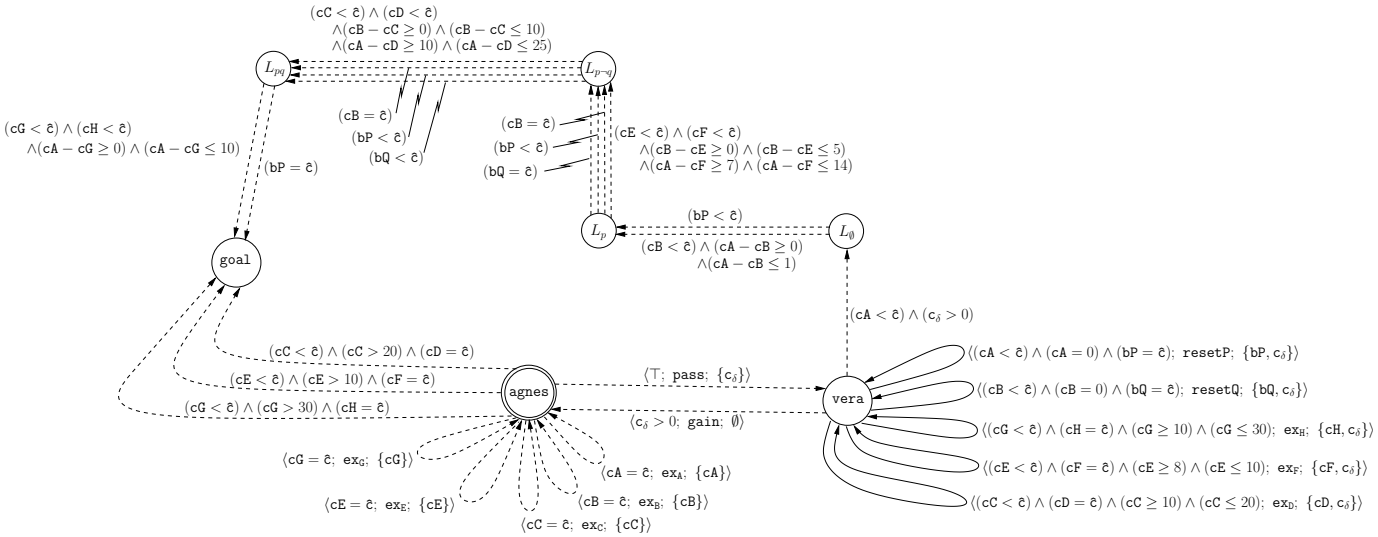


Fig. 8. The TGA derived from the sample CSTNU from Fig. 2

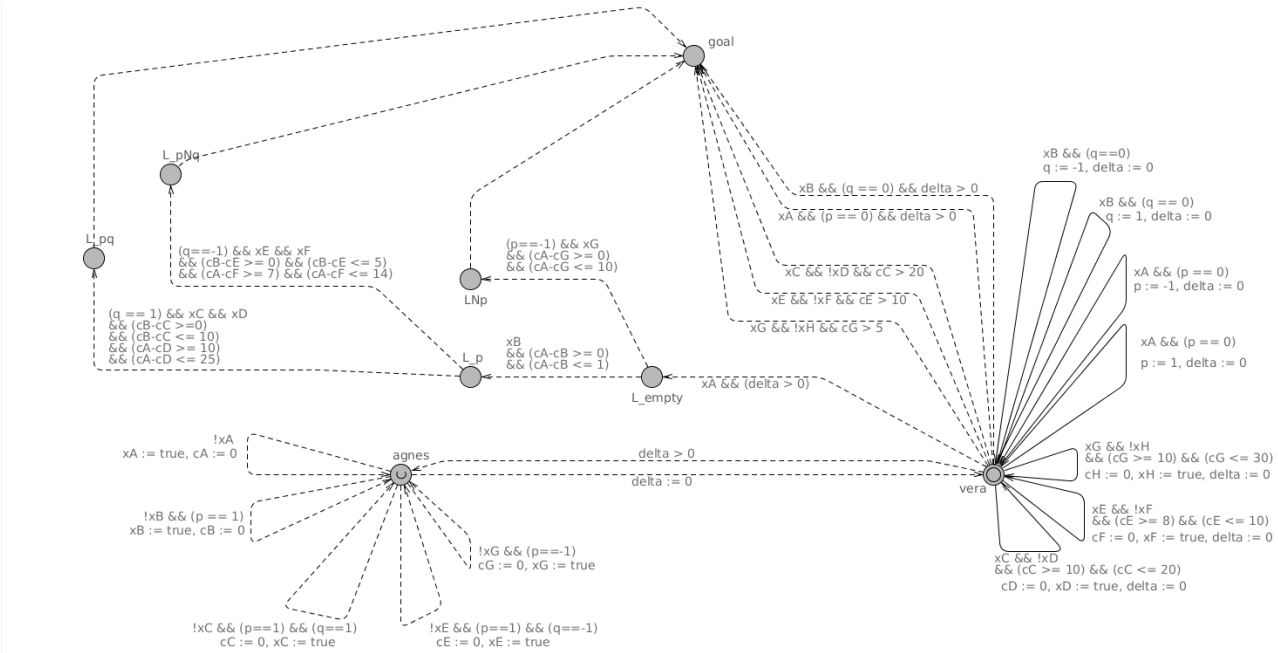


Fig. 9. An alternative TGA encoding of the CSTNU from Fig. 2, as shown by the UPPAAL-TIGA software [17]

$$\Sigma(\mathtt{cC}, \mathtt{cA}, \hat{\mathtt{c}}, l_i, u_i) \doteq (\mathtt{cA} < \hat{\mathtt{c}}) \wedge (\mathtt{cC} = \hat{\mathtt{c}}) \wedge (\mathtt{cA} \geq l_i) \wedge (\mathtt{cA} \leq u_i)$$

ensure that (one of) these transitions can be taken only when the link is currently activated and its duration would fall within one of the allowed intervals of $\mathcal{B}$. In addition, for each contingent constraint, there is a transition, $(\mathtt{agnes}, \Phi_C(\mathtt{cA}, \mathtt{cC}, \hat{\mathtt{c}}, max_i(u_i)), cv_C, \emptyset, \mathtt{goal})$ that allows Agnes to win the game if Vera refuses to schedule an uncontrollable time-point within the maximum allowed bound, $max_i(u_i)$. The guard is expressed by $\Phi_C(\mathtt{cA}, \mathtt{cC}, \hat{\mathtt{c}}, u) \doteq (\mathtt{cA} < \hat{\mathtt{c}}) \wedge (\mathtt{cA} > u) \wedge (\mathtt{cC} = \hat{\mathtt{c}})$, as in the STNU case. The interplay between the players, governed by the `pass` and `gain` transitions, is identical to the STNU case.

Next, the TGA must accommodate the arbitrary boolean combinations of constraints in $\mathcal{C}$. In principle, we would like to have a transition, $(\mathtt{agnes}, \Omega(\vec{c}, \hat{\mathtt{c}}), \mathtt{win}, \emptyset, \mathtt{goal})$ that signals the end of the game, where $\Omega(\vec{c}, \hat{\mathtt{c}})$ encodes the fact that all time-points have been executed and all constraints satisfied, and $\vec{c}$ is the set of clocks associated with the time-points.

First, let $\Lambda \doteq \bigwedge_{i=1}^{|\mathcal{C}|} \mathcal{C}_i$ be the first-order logic formula encoding all the constraints in $\mathcal{C}$. Then, let $\Omega(\vec{c}, \hat{\mathtt{c}})$ be the formula that results from replacing each atomic constraint, $X - Y \leq \delta$, with the equivalent clock constraint, $\mathtt{cY} - \mathtt{cX} \leq \delta$, while preserving the boolean structure of the formula:[10]

$$\Omega(\vec{c}, \hat{\mathtt{c}}) \doteq \Lambda[(X - Y \leq \delta)/(\mathtt{cY} - \mathtt{cX} \leq \delta)].$$

[10]Here we use the classical notation $\phi[x/y]$ for the substitution of the term x for the term y in the formula ϕ. However, we slightly abuse the notation by assuming that the substitution is applied to all atoms of the formula.

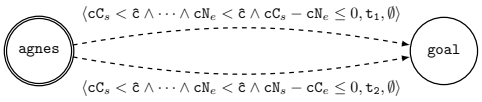


Fig. 10. The DNF encoding of the guards on constraints from agnes to goal for the sample DTNU

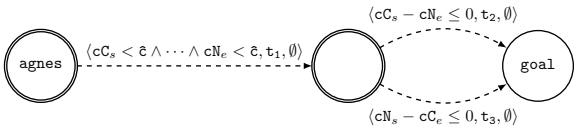


Fig. 11. NNF constraints between agnes and goal for the sample DTNU

For the DTNU from Fig. 1 that represents the non-overlapping cardiological and neurological evaluation tasks, Λ consists solely of the one constraint in $\mathcal{C}$:

$$\Lambda \doteq (N_e - C_s \leq 0) \vee (C_e - N_s \leq 0).$$

Therefore, $\Omega(\mathtt{cC}_s, \mathtt{cC}_e, \mathtt{cN}_s, \mathtt{cN}_e, \bar{\mathtt{c}})$ is equal to:

$$(\mathtt{cC}_s - \mathtt{cN}_e \leq 0) \vee (\mathtt{cN}_s - \mathtt{cC}_e \leq 0).$$

However, it is not always possible to directly use the formula $\Omega(\vec{c}, \bar{\mathtt{c}})$ as a guard for the transition from agnes to goal because the definition of a TGA restricts the language of the guards to be purely conjunctive. For this reason, we aim at building a piece of automaton—possibly adding new locations—that connects agnes to goal in such a way that the free constraints are equivalent to the disjunction of the conjunction of the guards along each path from agnes to goal. There are several ways in which this can be done.

Disjunctive Normal Form. Similarly to what has been done for CSTNUs, we can create a set of transitions from agnes to goal such that each pathway from agnes to goal can be taken if and only if $\Omega(\vec{c}, \bar{\mathtt{c}})$ is satisfied. This is always possible, since alternative transitions emanating from a single location are equivalent to a single transition with a disjunctive guard. Thus, all we have to do is convert $\Omega(\vec{c}, \bar{\mathtt{c}})$ into Disjunctive Normal Form (DNF) and create a separate transition from agnes to goal for every disjunct. In this setting, negation of atomic constraints is not a problem because $\neg(\mathtt{cY} - \mathtt{cX} \leq \delta)$ is equivalent to $\mathtt{cY} - \mathtt{cX} > \delta$, which is allowed in the guards of a TGA. As for the names of the actions, we assign to each action a unique new name. It is easy to see that there exists a path from agnes to goal if and only if the free constraints are satisfied, because one disjunct of the DNF is satisfied. The main drawback of this technique is that, for a general formula, the number of disjuncts in the DNF is exponential, and thus the encoding is exponential. Nevertheless, this constitutes a sound-and-complete encoding for (constructively) deciding the dynamic controllability of DTNU.

Considering again the running example, the encoding of the constraints between agnes and goal is composed of only two transitions, as Ω is already in DNF. The transitions are depicted in Fig. 10.

Negative Normal Form. If we allow for the introduction of new (urgent) locations in the TGA, we can encode $\Omega(\vec{c}, \bar{\mathtt{c}})$ linearly, thus obtaining a linear size of the overall DTNU-to-TGA encoding. The idea comes from the following observation. Suppose we have a piece of automaton that encodes a formula ϕ_1 in such a way that it is possible to move from location L_1^s to L_1^e if and only if ϕ_1 is satisfied, and suppose that we have an analogous encoding for another formula ϕ_2 with starting and ending locations L_2^s to L_2^e. We can encode the formula $\phi_1 \wedge \phi_2$ by "concatenating" the two automata. That is, we introduce a transition from L_1^e to L_2^s with the tautological guard $\top$. Now, in order to move from L_1^s to L_2^e the formula $\phi_1 \wedge \phi_2$ must be satisfied.[11] Similarly, if we consider the formula $\phi_1 \vee \phi_2$ we can introduce two extra locations $L_\vee^s$ and $L_\vee^e$ and introduce four transitions with the guard $\top$: one from $L_\vee^s$ to L_1^s, one from $L_\vee^s$ to L_2^s, one from L_1^e to $L_\vee^e$, and one from L_2^e to $L_\vee^e$. In this way, we create a "diamond" with two paths from $L_\vee^s$ to $L_\vee^e$; one path encodes ϕ_1, the other encodes ϕ_2. This construction is simple and correct, even though it introduces many unneeded locations. In fact it is also possible to compress this encoding by merging locations instead of linking them with tautological transitions and it is possible to merge sequences of guards in a single conjunctive guard. However, we decided to explain the simplest version for clarity.

It is easy to see that given an equivalent rewriting of $\Omega(\vec{c}, \bar{\mathtt{c}})$ that only has disjunctions and conjunctions (but no negations) we can recursively create a piece of automaton that encodes the formula. Now, we notice that we can syntactically and linearly transform $\Omega(\vec{c}, \bar{\mathtt{c}})$ into Negative Normal Form (NNF) and transform the negations of the atoms into positive atoms as before, by exploiting the fact that $\neg(\mathtt{cY} - \mathtt{cX} \leq \delta)$ is equivalent to $\mathtt{cY} - \mathtt{cX} > \delta$.

Fig. 11 depicts the running example encoded using the NNF decomposition of the free constraints optimized by compressing the conjunction of the $\mathtt{cX} < \bar{\mathtt{c}}$ in a single guard and by avoiding the unneeded tautological guards. In the running example, Ω is already in NNF as there are no negations.

Even though the NNF decomposition is always more succinct than the DNF, the effort of dealing with disjunctions is moved from the encoding to the TGA solver, so we are in a trade-off condition.

IV. Conclusions

The main contribution of this paper is to present, for the first time, a sound-and-complete algorithm for checking the dynamic controllability of temporal networks containing contingent links, observation time-points and disjunctive constraints—in any combination. The DC-checking algorithm is obtained by first translating the given temporal network into an equivalent Timed Game Automaton, and then using off-the-shelf software to synthesize a winning strategy—or confirm that no such strategy exists. Prior to this paper, this technique had only been applied to Simple Temporal Networks with Uncertainty, for which polynomial-time DC-checking algorithms already exist. The extension of this technique to temporal networks containing observation time-points and disjunctive constraints yields, for the first time, DC-checking algorithms for both CSTNUs and DTNUs.

[11] We are assuming that all the locations of the automata pieces are urgent, so the clocks are frozen and no time can elapse.

Because networks containing contingent links, observation time-points and disjunctive constraints can be used as the temporal foundation for a broad class of Workflow Management Systems currently being developed for automating business processes, the algorithmic work presented in this paper has immediate applications in industry, including in the healthcare domain.

There are many avenues for future work. First, although we have created an initial implementation of an automatic translator from CSTNUs to TGAs,[12] we have not yet carried out an extensive empirical evaluation. There are many different options for translating CSTNUs and DTNUs into TGAs; which options will generate the most efficient DC-checking algorithms is an open question.

REFERENCES

[1] R. Dechter, I. Meiri, and J. Pearl, "Temporal constraint networks," *Artificial Intelligence*, vol. 49, pp. 61–95, 1991.

[2] T. Vidal and H. Fargier, "Handling contingency in temporal constraint networks: from consistency to controllabilities," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 11, no. 1, pp. 23–45, 1999.

[3] P. H. Morris and N. Muscettola, "Temporal dynamic controllability revisited," in *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-2005)*, M. M. Veloso and S. Kambhampati, Eds. The MIT Press, 2005, pp. 1193–1198.

[4] P. Morris, "A structural characterization of temporal dynamic controllability," in *Principles and Practice of Constraint Programming (CP-2006)*, ser. Lecture Notes in Computer Science. Springer, 2006, vol. 4204, pp. 375–389.

[5] L. Hunsberger, "A faster algorithm for checking the dynamic controllability of simple temporal networks with uncertainty," in *Proceedings of the 6th International Conference on Agents and Artificial Intelligence (ICAART-2014)*, 2014.

[6] P. Morris, "Dynamic controllability and dispatchability relationships," in *Integration of AI and OR Techniques in Constraint Programming - 11th International Conference (CPAIOR-2014)*, ser. Lecture Notes in Computer Science, H. Simonis, Ed., vol. 8451. Springer, 2014, pp. 464–479.

[7] L. Hunsberger, "A fast incremental algorithm for managing the execution of dynamically controllable temporal networks," in *Proceedings of the 17th International Symposium on Temporal Representation and Reasoning (TIME-2010)*. Los Alamitos, CA, USA: IEEE Computer Society, 2010, pp. 121–128.

[8] ——, "A faster execution algorithm for dynamically controllable stnus," in *Proceedings of the 20th International Symposium on Temporal Representation and Reasoning (TIME-2013)*, C. Sánchez, K. B. Venable, and E. Zimányi, Eds. IEEE, 2013, pp. 26–33.

[9] C. Combi, M. Gambini, S. Migliorini, and R. Posenato, "Representing business processes through a temporal data-centric workflow modeling language: An application to the management of clinical pathways," *IEEE Transactions on Systems, Man, and Cybernetics*, 2014. [Online]. Available: http://dx.doi.org/10.1109/TSMC.2014.2300055

[10] L. Hunsberger, R. Posenato, and C. Combi, "The Dynamic Controllability of Conditional STNs with Uncertainty," in *Proceedings of the Workshop on Planning and Plan Execution for Real-World Systems: Principles and Practices (PlanEx) at ICAPS-2012*, 2012, pp. 1–8.

[11] C. Combi, L. Hunsberger, and R. Posenato, "An algorithm for checking the dynamic controllability of a conditional simple temporal network with uncertainty," in *Proceedings of the 5th International Conference on Agents and Artificial Intelligence (ICAART-2013)*, J. Filipe and A. L. N. Fred, Eds. SciTePress, 2013, pp. 144–156.

[12] K. B. Venable, M. Volpato, B. Peintner, and N. Yorke-Smith, "Weak and dynamic controllability of temporal problems with disjunctions and uncertainty," in *Proceedings of the Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS-2010) in ICAPS-2010*, 2010, pp. 50–59.

[13] B. Peintner, K. B. Venable, and N. Yorke-Smith, "Strong controllability of disjunctive temporal problems with uncertainty," in *Principles and Practice of Constraint Programming (CP-2007)*, 2007, pp. 856–863.

[14] A. Cimatti, A. Micheli, and M. Roveri, "Solving temporal problems using SMT: strong controllability," in *Principles and Practice of Constraint Programming (CP-2012)*, 2012, pp. 248–264.

[15] A. Cimatti, L. Hunsberger, A. Micheli, and M. Roveri, "Using timed game automata to synthesize execution strategies for simple temporal networks with uncertainty," in *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI-2014)*, 2014.

[16] O. Maler, A. Pnueli, and J. Sifakis, "On the synthesis of discrete controllers for timed systems," in *Proceedings of the 12th Symposium on Theoretical Aspects of Computer Science (STACS-1995)*, 1995, pp. 229–242.

[17] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. Larsen, and D. Lime, "Uppaal-Tiga: Time for playing games!" in *Proceedings of the 19th Conference on Computer Aided Verification (CAV-2007)*, ser. Lecture Notes in Computer Science, W. Damm and H. Hermanns, Eds. Springer Berlin Heidelberg, 2007, vol. 4590, pp. 121–125.

[18] D. Hollingsworth, "The workflow reference model," http://www.wfmc.org/standards/model.htm, 1995.

[19] J. Eder, E. Panagos, and M. Rabinovich, "Time constraints in workflow systems," in *Advanced Information Systems Engineering*, ser. LNCS, M. Jarke and A. Oberweis, Eds. Springer Berlin Heidelberg, 1999, vol. 1626, pp. 286–300.

[20] C. Combi and G. Pozzi, "Architectures for a temporal workflow management system," in *Proceedings of the 2004 ACM Symposium on Applied Computing (SAC-2004)*. New York, NY, USA: ACM, 2004, pp. 659–666.

[21] C. Combi, M. Gozzi, R. Posenato, and G. Pozzi, "Conceptual modeling of flexible temporal workflows," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 7, no. 2, pp. 19:1–19:29, 2012.

[22] C. Combi and R. Posenato, "Controllability in temporal conceptual workflow schemata," in *Business Process Management*, ser. LNCS, U. Dayal, J. Eder, J. Koehler, and H. Reijers, Eds. Springer Berlin Heidelberg, 2009, vol. 5701, pp. 64–79.

[23] P. Morris, N. Muscettola, and T. Vidal, "Dynamic control of plans with temporal uncertainty," in *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-2001)*, B. Nebel, Ed. Morgan Kaufmann, 2001, pp. 494–499.

[24] I. Tsamardinos, T. Vidal, and M. Pollack, "Ctp: A new constraint-based formalism for conditional, temporal planning," *Constraints*, vol. 8, no. 4, pp. 365–388, 2003.

[25] K. B. Venable and N. Yorke-Smith, "Disjunctive temporal planning with uncertainty," in *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-2005)*, 2005, pp. 1721–1722.

[26] I. Tsamardinos and M. E. Pollack, "Efficient solution techniques for disjunctive temporal reasoning problems," *Artificial Intelligence*, vol. 151, pp. 43–89, 2003.

[27] F. Cassez, "Efficient on-the-fly algorithms for partially observable timed games," in *Proceedings of the 5th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS-2007)*, 2007, pp. 5–24.

[28] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994.

[12] An initial implementation of our translator is available at: http://profs.sci.univr.it/~posenato/software/cstnu.