A Sound-and-Complete Propagation-based Algorithm for Checking the Dynamic Consistency of Conditional Simple Temporal Networks

Luke Hunsberger
Vassar College
Poughkeepsie, NY USA

Roberto Posenato Carlo Combi
Computer Science Department
University of Verona, Verona, Italy

***Abstract*—A Conditional Simple Temporal Network (CSTN) is a data structure for representing and reasoning about time-points and temporal constraints, some of which may apply only in certain scenarios. The scenarios in a CSTN are represented by conjunctions of propositional literals whose truth values are not known in advance, but instead are observed in real time, during execution. The most important property of a CSTN is whether it is dynamically consistent (DC); that is, whether there exists a strategy for executing its time-points such that all relevant constraints are guaranteed to be satisfied no matter which scenario is incrementally revealed during execution. Prior approaches to determining the dynamic consistency of CSTNs (a.k.a., solving the Conditional Simple Temporal Problem) are primarily of theoretical interest; they have not been realized in practical algorithms. This paper presents a sound-and-complete DC-checking algorithm for CSTNs that is based on the propagation of constraints labeled by propositions. The paper also presents an empirical evaluation of the new algorithm that demonstrates that it may be practical for a variety of applications. This is the first empirical evaluation of any DC-checking algorithm for CSTNs ever reported in the literature.**

***Index Terms*—Constraint-based Temporal Reasoning; Temporal Networks; Constraint Satisfaction; Scheduling.**

I. Overview

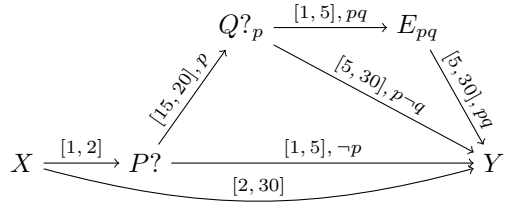
A Conditional Simple Temporal Network (CSTN) is a data structure for representing and reasoning about temporal constraints in domains where some constraints may apply only in certain scenarios. For example, a patient who tests positive for a certain disease may need to receive care more urgently than a patient who tests negative. Each condition in a CSTN is represented by a propositional letter whose truth value is not controlled, but instead is *observed* in real time. Just as the performance of a blood-test action by a doctor might generate a positive or negative result that is only learned in real time, the execution of an *observation time-point* in a CSTN generates, in real time, a truth value for its corresponding propositional letter. 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 that 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.

Tsamardinos *et al.* [1] introduced the CSTP. They solved it by encoding it as a meta-level Disjunctive Temporal Network (DTN), then feeding it to 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. To our knowledge, this approach has never been implemented or empirically evaluated.

More recently, Cimatti *et al.* [2] presented a novel approach to solving a variety of temporal problems in which a temporal network is first translated into an equivalent Timed Game Automaton (TGA) before being solved by an off-the-shelf TGA solver. This approach illuminates the fascinating relationships between TGAs and a variety of temporal networks—including CSTNs. However, it remains to be seen whether this approach can yield practical DC-checking algorithms.

This paper presents a new approach to solving the DC-checking problem for CSTNs: one based on constraint propagation. The approach draws inspiration from two different lines of research from the literature. First, like Conrad and Williams [3], our approach labels constraints with propositions. However, whereas Conrad and Williams use propositional labels to represent alternative scenarios from among which an agent can *choose,* our approach uses propositional labels to represent the scenarios in a CSTN that are incrementally *revealed* and, thus, not controlled by the planning agent. Second, the CSTN semantics and *some* of the constraint-propagation rules in this paper draw from prior work on Conditional Simple Temporal Networks with Uncertainty (CSTNUs) [4]–[6]. That work presents a variety of sound constraint-propagation rules in the more general CSTNU setting; however, it has not yet provided a *complete* set of rules and, therefore, has not yet resulted in a sound-and-complete dynamic-*controllability* algorithm for CSTNUs.[1] Our approach applies some of their constraint-propagation rules in the more restrictive CSTN setting, but augments them to include additional rules that guarantee

[1]The DC-checking problem for CSTNs is quite different from the DC-checking problem for STNUs [7] or CSTNUs [4]. For STNUs and CSTNUs, which contain uncertain durations called *contingent links,* the "C" in DC-checking stands for "controllability".



Time-Point	Meaning
X	Begin
$P?$	Do Blood Test (generates truth value for p; $p = true$ indicates positive reaction)
$Q?_p$	Repeat Blood Test (only applicable if $p = true$; generates truth value for q; $q = true$ indicates positive reaction)
E_{pq}	Do Emergency Procedure (only applicable if $p = q = true$)
Y	End

Fig. 1: A simple CSTN in a health-care setting.

completeness. As a result, our approach provides the first sound-*and-complete* DC-checking algorithm for CSTNs that is based on constraint propagation. The paper empirically evaluates the new algorithm, demonstrating its practicality across a variety of networks. Future work will explore the possibility of extending the novel techniques from this paper to the more general problem of checking the dynamic controllability of CSTNUs, for which no *practical* sound-and-complete algorithm yet exists.

II. Background

Dechter *et al.* [8] introduced Simple Temporal Networks (STNs) to facilitate representing and reasoning about temporal constraints. An STN comprises real-valued variables, called *time-points,* and binary difference constraints on those variables. The *Simple Temporal Problem* (STP) is the problem of determining whether an STN is consistent (i.e., has a solution).

Tsamardinos *et al.* [1] augmented STNs to include time-points and temporal constraints that apply only in certain scenarios, where each scenario is represented by a conjunction of propositional literals.[2] For convenience, in this paper, conjunctions such as $p \wedge \neg q \wedge r$ are frequently notated as $p\neg qr$. Fig. 1 provides a simple example of a CSTN in a health-care setting. In the figure, the CSTN is shown in its graphical form, where the nodes $X, P?, Q?, E$ and Y represent time-points, and the directed edges represent binary difference constraints. For example, the time-point $P?$ represents the time at which a particular blood test is performed. This test generates a truth value for the propositional letter p, where $p = true$ indicates that the patient tested positive for a particular reagent. In this example, if the test generates a positive result, then the test is repeated at time-point $Q?$, which generates a truth value for the propositional letter q. Since the time-point $Q?$ applies only in scenarios where $p = true$, it is labeled by p. Similarly, the edge from $P?$ to $Q?$ is labeled by p. It represents the constraint, $Q? - P? \in [15, 20]$ (i.e., the repeated test must be performed between 15 and 20 minutes after the first test). Finally, note that the constraints from $Q?$ to E, and from E to Y are labeled by pq, indicating that they apply only in scenarios where both p and q are $true$.

The following definitions are equivalent to those from Tsamardinos *et al.* [1], except that they explicitly attach labels to constraints as well as time-points, as in Hunsberger *et al.* [4].

Definition 1 (Labels)**.** Given a set $\mathcal{P}$ of propositional letters:

- a *label* is a (possibly empty) conjunction of (positive or negative) literals from $\mathcal{P}$. The empty label is notated $\boxdot$.
- for any label ℓ, and any $p \in \mathcal{P}$, if $\ell \models p$ or $\ell \models \neg p$, then we say that p *appears* in ℓ.
- for any labels, ℓ_1 and ℓ_2, if $\ell_1 \models \ell_2$ (i.e., if ℓ_1 contains all of the literals in ℓ_2) then ℓ_1 is said to *entail* ℓ_2.[3] If $\ell_1 \wedge \ell_2$ is satisfiable, then ℓ_1 and ℓ_2 are called *consistent.*
- the *label universe* of $\mathcal{P}$, denoted by $\mathcal{P}^*$, is the set of all *consistent* labels whose literals are drawn from $\mathcal{P}$.

Definition 2 (CSTN)**.** A *Conditional Simple Temporal Network* (CSTN) is a tuple, $\langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, \mathcal{P} \rangle$, where:

- $\mathcal{P}$ is a finite set of propositional letters (or propositions);
- $\mathcal{T}$ is a finite set of real-valued time-points (i.e., variables);
- $\mathcal{C}$ is a set of *labeled* constraints, each having the form, $(Y - X \leq \delta,\ \ell)$, where $X, Y \in \mathcal{T}$, $\delta \in \mathbb{R}$, and $\ell \in \mathcal{P}^*$;
- $L : \mathcal{T} \rightarrow \mathcal{P}^*$ is a function assigning labels to time-points;
- $\mathcal{OT} \subseteq \mathcal{T}$ is a (finite) set of observation time-points; and
- $\mathcal{O} : \mathcal{P} \rightarrow \mathcal{OT}$ is a bijection that associates a unique observation time-point to each propositional letter.

In a CSTN graph, $\mathcal{O}(p)$ (i.e., the observation time-point associated with p) may be denoted by $P?$; and each labeled constraint, $(Y - X \leq \delta, \ell)$, is represented by an arrow from X to Y annotated by the *labeled value,* $\langle \delta, \ell \rangle$.[4] Any time-points, X and Y, may participate in multiple constraints of the form, $(Y - X \leq \delta_i, \ell_i)$; thus, the edge from X to Y in the CSTN graph may have multiple labeled values of the form, $\langle \delta_i, \ell_i \rangle$.

In addition to the requirements listed in Defn. 2, Tsamardinos *et al.* [1] asserted that well-defined CSTNs should exhibit certain additional properties. For example, if a time-point X applies only in scenarios where p is $true$ (i.e., if the label on the time-point X includes p), then X should not be executed before the truth value of p has been observed (i.e., X should be constrained to occur after the corresponding observation time-point $P?$). Later work [4] collected such *well-definedness* properties, calling them WD_1, WD_2 and WD_3. Below, the terms *label honesty* and *label coherence* are introduced to facilitate the presentation of the WD properties.

Honest Labels

Suppose that the label for an observation time-point $Q?$ contains the literal p, indicating that $Q?$ applies only in scenarios where

[2] Tsamardinos et al. only attached labels to time-points. They left it implicit that constraints among labeled time-points may apply only in certain scenarios.

[3] In Tsamardinos et al., ℓ_1 was said to *subsume* ℓ_2. However, that use of the term, subsumption, conflicts with the standard usage of that term, whereby a subsumer is more general than its subsumee [9].

[4] The graph in Fig. 1 follows an alternative convention, whereby the constraint, $(Y - X \in [a, b], \ell)$, is represented by an arrow from X to Y annotated by $[a, b], \ell$. This is equivalent to the pair of constraints, $(Y - X \leq b, \ell)$ and $(X - Y \leq -a, \ell)$.

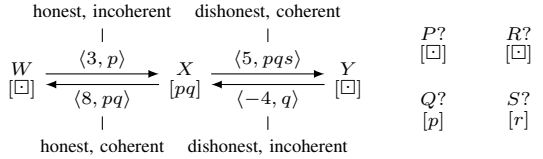


Fig. 2: Examples of label honesty and coherence

p is $true$; and that the label for another time-point E contains the literal q, indicating that E applies only in scenarios where q is $true$. But then E applies only in scenarios where $Q?$ has been executed—and, hence, where p is $true$. Thus, if the label on E is *honest,* it should include both p and q, as shown in Fig. 1. More generally, the label on E should include all of the literals contained in the label on $Q?$ (i.e., the label on E should *entail* the label on $Q?$). And these remarks also apply to labels on constraints, as follows.

Definition 3 (Honest Label)**.** A label ℓ in a CSTN, whether on a time-point or constraint, is called *honest* if for each q that appears in ℓ, ℓ entails $L(Q?)$ (i.e., ℓ contains all literals from the label of the observation time-point for q).

An honest label is equivalent to a *minimum execution scenario* for a CSTP [1].

Coherent Labels

Suppose that a time-point X applies only in scenarios where p is $true$, and another time-point Y applies only in scenarios where q is $false$. Then any constraint involving both X and Y applies only in scenarios where p is $true$ and q is $false$. Thus, to be *coherent,* the label on such a constraint should include both p and $\neg q$.

Definition 4 (WD_1: Label coherence)**.** A CSTN holds property WD_1 (i.e., has *coherent labels*) if for each labeled constraint, $(Y - X \leq \delta, \ell)$, the label ℓ is satisfiable and entails $L(X) \wedge L(Y)$ (i.e., contains all literals from $L(X)$ and $L(Y)$).

It is easy to confirm that all of the labels in Fig. 1 are honest, and all of the labels on constraints are coherent. In contrast, the graph in Figure 2 contains labels that exhibit different combinations of label honesty and coherence. In the figure, the labels on time-points are enclosed in square brackets.

Definition 5 (WD_2:)**.** A CSTN holds property WD_2 if for each time-point $T \in \mathcal{T}$, its label $L(T)$ is *honest,* and for each $p \in \mathcal{P}$ that appears in $L(T)$, $(\mathcal{O}(p) - T \leq -\epsilon, L(T)) \in \mathcal{C}$, for some $\epsilon > 0$.

Definition 6 (WD_3: Constraint Label Honesty)**.** A CSTN holds property WD_3 if the label on each of its constraints is *honest.*

In brief, WD_1 ensures that labels on constraints entail the labels on the corresponding endpoints; WD_2 ensures that time-point labels are honest, and that each time-point is constrained to occur after the observation time-points corresponding to propositions appearing in its label; and WD_3 ensures that edge labels are honest.

Definition 7 (Well-Defined CSTN)**.** A CSTN is said to be *well defined* if properties WD_1, WD_2 and WD_3 hold for it.

Note. The sample CSTN from Fig. 1 is well defined. The rest of the paper restricts attention to well-defined CSTNs.

A. The Dynamic Consistency of CSTNs

The truth values of propositions in a CSTN are not known in advance; instead, they are incrementally revealed over time as the observation time-points execute. A *strategy* for executing the time-points in a CSTN is defined in a way that allows it to react to observations in real time. For example, suppose that the observation time-point $P?$ in the CSTN from Fig. 1 is executed at time 2, revealing the truth value of p. Subsequent execution decisions may react to that observation. For example, if $p = true$, then $Q?$ might be executed next, say, at time 18; whereas if $p = false$, then Y might be executed next, say, at time 4. In any case, execution decisions must not be allowed to depend on advance knowledge of future events.

A *viable* and *dynamic* execution strategy for a CSTN is a strategy that guarantees that all relevant constraints will be satisfied no matter which scenario is incrementally revealed over time. A CSTN with such a strategy is called *dynamically consistent.* The semantics of dynamic consistency is given by the following definitions, drawn from Hunsberger *et al.* [4].

Definition 8 (Scenario/Interpretation Function)**.** A *scenario* (or *interpretation function*) over a set $\mathcal{P}$ of propositional letters is a function, $s : \mathcal{P} \rightarrow \{true, false\}$, that assigns a truth value to each letter in $\mathcal{P}$. Any such function also provides the truth value for any label $\ell \in \mathcal{P}^*$, which is denoted by $s(\ell)$. The set of all scenarios over $\mathcal{P}$ is denoted by $\mathcal{I}_{\mathcal{P}}$ (or simply $\mathcal{I}$).

Definition 9 (Schedule)**.** A *schedule* for a set of time-points $\mathcal{T}$ is a mapping, $\psi : \mathcal{T} \rightarrow \mathbb{R}$, that assigns a real number to each time-point in $\mathcal{T}$. The set of all schedules *for any subset of* $\mathcal{T}$ is denoted by $\Psi_{\mathcal{T}}$ (or simply Ψ, if context allows).

The projection of a CSTN, $\mathcal{S}$, onto a scenario, s, is the STN obtained by collecting all time-points and constraints from $\mathcal{S}$ whose labels are true under s (i.e., the time-points that must be executed and the constraints that must be satisfied under s).

Definition 10 (Projection)**.** Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, \mathcal{P} \rangle$ be any CSTN, and s any scenario over $\mathcal{P}$. The *projection* of $\mathcal{S}$ onto s—notated $scPrj(\mathcal{S}, s)$—is the STN, $(\mathcal{T}_s^+, \mathcal{C}_s^+)$, where:

- $\mathcal{T}_s^+ = \{T \in \mathcal{T} \mid s(L(\mathcal{T})) = true\}$; and
- $\mathcal{C}_s^+ = \{(Y - X \leq \delta) \mid$ for some $\ell, (Y - X \leq \delta, \ell) \in \mathcal{C}$ and $s(\ell) = true\}$

Definition 11 (Execution Strategy)**.** An *execution strategy* for a CSTN $\mathcal{S} = \langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, \mathcal{P} \rangle$ is a mapping, $\sigma : \mathcal{I} \rightarrow \Psi_{\mathcal{T}}$, such that for each scenario $s \in \mathcal{I}$, the domain of $\sigma(s)$ is $\mathcal{T}_s^+$ (cf. Defn. 10). If, in addition, for each scenario s, the schedule $\sigma(s)$ is a solution to the projection $scPrj(\mathcal{S}, s)$, then σ is called *viable.* In any case, the execution time for the time-point X in the schedule $\sigma(s)$ is denoted by $[\sigma(s)]_X$.

Up to this point, there is nothing in the definition of an execution strategy that prevents it from making decisions based

on advance knowledge of future observations. The following definitions ensure that a *dynamic execution strategy* makes decisions that depend only on past observations. For any time t, the observations before time t are recorded in the *history.*

Definition 12 (History)**.** Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, \mathcal{P} \rangle$ be any CSTN, s any scenario, σ any execution strategy for $\mathcal{S}$, and t any real number. The *history* of t in the scenario s, for the strategy σ—notated $Hist(t, s, \sigma)$—is the set of observations made before time t according to the schedule $\sigma(s)$:

$$Hist(t, s, \sigma) = \{(p, s(p)) \mid \mathcal{O}(p) \in \mathcal{T}_s^+ \text{ and } [\sigma(s)]_{\mathcal{O}(p)} < t\}$$

Definition 13 (Dynamic Execution Strategy)**.** An execution strategy σ for a CSTN is called *dynamic* if for any scenarios s_1 and s_2, and any time-point X:

let: $t = [\sigma(s_1)]_X$
if: $Hist(t, s_1, \sigma) = Hist(t, s_2, \sigma)$
then: $[\sigma(s_2)]_X = t$.

In other words, if a dynamic execution strategy σ executes X at time t in scenario s_1, and the schedules $\sigma(s_1)$ and $\sigma(s_2)$ have the same history of observations up to time t, then σ must execute X at that same time t in scenario s_2. In this way, the decisions of σ can only depend on past observations.

Definition 14 (Dynamic Consistency)**.** A CSTN $\mathcal{S}$ is *dynamically consistent* (DC) if there exists an execution strategy for it that is both dynamic and viable.

The dynamic property ensures that the execution decisions made by the strategy depend only on past observations; viability ensures that all relevant constraints will be satisfied no matter which scenario is incrementally revealed over time.

III. DC-Checking for CSTNs based on the Propagation of Labeled Constraints

Algorithms for checking the dynamic consistency of arbitrary CSTNs are called *DC-checking* algorithms. As already mentioned, sound and complete DC-checking algorithms for CSTNs have already been presented [1], [2]. However, they are primarily of theoretical interest; the algorithms either have not been implemented or only work on small networks.

The algorithm presented in this paper follows a different approach, one that is based on the propagation of *labeled* constraints. Although this kind of approach has been applied to the more general problem of checking the dynamic *controllability* of CSTNUs [4], it has only met with limited success in that context; to date, it has not yielded a sound-and-*complete* algorithm. This paper presents a new set of constraint-propagation rules that draws from the prior work on CSTNUs, but includes new rules that take into account important features in CSTN graphs (e.g., *negative q-loops* and *negative q-stars*) that have never been identified before. The result is a DC-checking algorithm for CSTNs that is both sound *and complete.* In addition, the empirical evaluation of our initial implementation suggests that, unlike its predecessors, the new algorithm may be practical for a wide variety of applications.

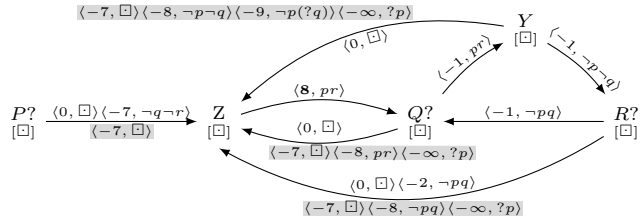


Fig. 3: A dynamically consistent CSTN.

A. Motivation

Figure 3 shows a CSTN with three observation time-points—$P?$, $Q?$ and $R?$—and two other time-points, Y and Z. As in many STNs, Z is a special starting point whose value is fixed at zero. In the figure, the shaded labeled values, some of which include novel annotations, are generated by constraint propagation, to be discussed later. The labeled values on edges terminating at Z are used by the earliest-first execution strategy, discussed below.

1) Lower bounds and the current partial scenario: To ensure that each time-point X cannot be executed before Z in any scenario, it suffices to include a labeled constraint of the form, $(\mathrm{Z} - X \leq 0, \boxdot)$, which, since $\mathrm{Z} = 0$, may be more conveniently notated as $(X \geq 0, \boxdot)$. In the CSTN graph, such a constraint is represented by an edge from X to Z labeled by $\langle 0, \boxdot \rangle$. More generally, for each X, any labeled constraint of the form, $(\mathrm{Z} - X \leq -\delta, \ell)$, represents a lower-bound constraint that applies in all scenarios that are consistent with the label ℓ. Such a constraint, which may be more conveniently notated as $(X \geq \delta, \ell)$, is represented by an edge in the CSTN graph from X to Z labeled by $\langle -\delta, \ell \rangle$.

During execution, as observation time-points are executed, truth values for the corresponding propositional letters are generated. Each observation has the potential to make a large number of labeled constraints irrelevant. For example, if p is observed to be $true$, then any constraint whose label includes $\neg p$ need not be satisfied and, thus, can thenceforth be ignored. As a result, such observations can change the effective lower bounds for any as-yet-unexecuted time-points. The computation of such bounds depends on the observations that have occurred so far, which can be concisely represented by a label, called the *current partial scenario* (CPS). For example, before any observations have been made, the CPS is represented by the empty label, $\boxdot$. Later on, if $P?$ generated the result $p = true$, then the CPS would be updated to p. Still later, if $Q?$ generated $q = false$, then the CPS would be updated to $p\neg q$. And so on.

Let ℓ be a current partial scenario, and X an as-yet-unexecuted time-point whose label $L(X)$ is consistent with ℓ. If $\ell \not\models L(X)$, then for some $p \in L(X)$, the observation time-point $P?$ has not yet been executed and, since X must occur after $P?$, it follows that X cannot be executed *next* by any viable strategy. However, if $\ell \models L(X)$, then X could be executed next; and the *effective lower bound* (ELB) for X can be defined as the maximum among the lower bounds for X whose labels are consistent with ℓ.

2) The earliest-first strategy: Once a CSTN is known to be DC, a wide variety of dynamic execution strategies may be available for successfully executing the time-points in the network. However, the proof of completeness for the DC-checking algorithm presented in this paper need only demonstrate the existence of one such strategy. For this purpose, an iterative *earliest-first* execution strategy is defined. At the beginning of each iteration, the earliest-first strategy computes the effective lower bound for each unexecuted time-point whose label is entailed by the current partial scenario. It then identifies the time-point X^* whose effective lower bound is minimal, and executes X^* at that minimal lower bound (i.e., as early as possible). If X^* is an observation time-point, the strategy updates the current partial scenario accordingly, in preparation for the next iteration. The process continues until all time-points have been executed. As will be seen, correctly computing the relevant lower bounds requires greater care than has been so far indicated.

3) Instantaneous reaction: Recall that the definitions of histories and dynamic execution strategies (Defns. 12 and 13) require execution decisions to depend only on *past* observations, not on present or future observations. However, to simplify the *earliest-first* strategy used in this paper, we allow it to make decisions based on past *or present* observations. In other words, it will be allowed to *react instantaneously* to the truth values that are generated by the execution of observation time-points. As has been noted in similar contexts [10], this kind of simplification does not affect the basic results, but it enables avoiding situations where there is no "earliest" time (e.g., in the open interval immediately following an observation). In this context, it suffices to replace the original definition of dynamic execution strategy (cf. Defn. 13) with that of an *IR-dynamic execution strategy,* given below. ("IR" stands for "Instantaneous Reaction".) Note that the only difference between Defns. 13 and 15 is the "unless" clause in Defn. 15.

Definition 15 (IR-Dynamic Execution Strategy)**.** An execution strategy σ for a CSTN is called *IR-dynamic* if for any scenarios s_1 and s_2, and any time-point X:

let: $t = [\sigma(s_1)]_X$
if: $\mathit{Hist}(t, s_1, \sigma) = \mathit{Hist}(t, s_2, \sigma)$
then: $[\sigma(s_2)]_X = t$
unless: $[\sigma(s_1)]_{P?} = [\sigma(s_2)]_{P?} = t$ and $s_1(p) \neq s_2(p)$ for some $p \in P^*$, where $X \not\equiv P?$.

Thus, if the two schedules, $\sigma(s_1)$ and $\sigma(s_2)$, both execute the observation time-point $P?$ at time t, but generate different truth values for p, then X need not be executed at time t in the schedule $\sigma(s_2)$. That is, σ may instantaneously react to the observed value of p in scenario s_2 by choosing to execute X at some later time.

For convenience, in the rest of this paper, the term *dynamic execution strategy* is used to refer to *IR-dynamic execution strategies.*

It is not hard to verify that the CSTN in Figure 3 is DC. Indeed, Algorithm 1 provides an execution strategy that dynamically generates schedules that are guaranteed to satisfy all of the constraints in the network no matter how the observations of p, q and r turn out. This strategy employs instantaneous reaction. For example, if executing $P?$ at 7 generates $p = \mathit{true}$, then $R?$ is immediately executed, also at time 7. Later sections show how this strategy can be generated.

Algorithm 1: Execution strategy for the CSTN from Fig. 3

```
Execute Z at 0, and P? at 7;
if (p == true) then
    Execute both R? and Y at 7;
    if (r == false) then Execute Q? at 7;
    else Execute Q? at 8 ;
else
    Execute Q? at 7;
    if (q == true) then Execute Y at 7, and R? at 8;
    else Execute R? at 7, and Y at 8 ;
```

Since a CSTN is a restricted form of CSTNU, the constraint-propagation rules for CSTNUs [5], [6] can also be applied to CSTNs. However, those rules—hereinafter called the CSTNU rules—are not sufficient for solving the DC-checking problem for arbitrary CSTNs, let alone the analogous problem for CSTNUs. The main reason is that the CSTNU rules only propagate constraints when consecutive edges have mutually consistent labels which turns out to be too restrictive, as discussed below.

4) Negative q-loops: Consider the loop from $Q?$ to Y to $R?$ to $Q?$ in Figure 3 whose edges have labels that are pairwise inconsistent (e.g., pr is inconsistent with $\neg p \neg q$, which is inconsistent with $\neg pq$). Now, as long as the values of p, q and r are all unknown, it is not safe to violate any of those constraints; however, since the loop has negative length, no schedule can jointly satisfy them all. Because such loops are only problematic when one or more propositional letters have unknown truth values, we call such loops *negative q-loops,* where the "q" stands for "question mark". For the sample network, the conundrum presented by the negative q-loop can be resolved by first executing $P?$, observing the result, and then reacting appropriately. For example, if executing $P?$ yields $p = \mathit{false}$, then the constraint from $Q?$ to Y can be ignored, effectively resolving the negative q-loop. The DC-checking algorithm for CSTNs presented in the next section properly addresses the challenges raised by negative q-loops.

5) Negative q-stars: Another problem arises when computing the effective lower bounds for unexecuted time-points, needed during execution. Recall that the ELB for each time-point X derives from the strongest labeled value on the edge from X to Z that is consistent with the current partial scenario. But restricting constraint propagation to mutually consistent labels turns out to be insufficient to generate all such labeled values. For example, consider the labeled values, $\langle -1, \neg p \neg q \rangle$ and $\langle -2, \neg pq \rangle$, that appear on the consecutive edges from Y to $R?$ to Z in Fig. 3. Prior to any observations, both of the corresponding constraints must be satisfied, despite their mutually inconsistent labels. The question is: what should be

the label on the generated lower-bound edge from Y to Z?

Since the lower-bound edges from unexecuted time-points all point toward Z, and typically have negative lengths, we call any collection of such edges a *negative q-star*. For example, consider the negative q-star formed by the lower-bound edges for $P?, Q?, R?$ and Y in Figure 3. Before any propagation, the respective lower bounds for these time-points are $7, 0, 2$ and 0. Propagating constraints across consistent labels yields only one update: the path from $Q?$ to Y to Z generates a new edge (not shown in the figure) from $Q?$ to Z whose labeled value is $\langle -1, pr \rangle$, representing a lower bound of 1 for $Q?$ in scenarios where p and r are $true$. However, as will be seen in the next section, appropriate propagation of constraints across inconsistent labels in this negative q-star generates much stronger lower bounds for these time-points: 7 for each one, as indicated by the shaded labeled values, $\langle -7, \boxdot \rangle$, in the figure.

The insufficiency of the CSTNU rules to properly address constraint propagation becomes even more dramatic if the labeled value on the edge from Z to Q is changed from $\langle 8, pr \rangle$ to $\langle 7, pr \rangle$. In that case, the network is no longer DC, but the existing CSTNU rules are unable to detect it. Since they ignore mutually inconsistent labels, they are unable to determine that $Q?$ must be at least 8 after Z in the case of pr and, therefore, miss the (unresolvable) negative cycle between Z and $Q?$.

B. New Constraint-Propagation Rules for CSTNs

As already mentioned, prior work has provided a variety of sound-*but-not-complete* constraint-propagation rules for the more general case of CSTNUs [5], [6]. Our approach begins with a small subset of those rules, restricting them to CSTNs and modifying them slightly to correct some oversights. It then introduces novel variants of the rules to handle problems raised by negative q-loops and negative q-stars. The resulting set of CSTN rules is then shown to be both sound and complete for the purposes of DC-checking for CSTNs.

To facilitate the presentation of the constraint-propagation rules for CSTNs, *child nodes* are introduced. To illustrate the concept, recall the CSTN from Fig. 1 in which $Q?$ is an observation time-point whose label contains the literal p. Thus $Q?$ only applies in scenarios where p is $true$. In such cases, $Q?$ is called a *child* of $P?$. (We may also say that q is a child of p.) Note that if ℓ is an honest label that contains q, it must also contain p. Conversely, an honest label that does not contain p cannot contain q or any other child of p.

Definition 16. If $Q?$ is an observation time-point whose label contains an occurrence of p (resp., $\neg p$), then $Q?$ is called a *child* of $P?$. We may also say that q is a child of p (resp., $\neg p$).

Using child nodes enables the constraint-propagation rules to be specified in a way that ensures that they preserve label honesty, an issue that was not addressed in prior work.

Note. In a well-defined CSTN, if p appears in the label $L(X)$, then X is constrained to occur after $P?$. Similarly, if $Q?$ is a child of $P?$, then $Q?$ is constrained to occur after $P?$. Hence, the current partial scenario resulting from any viable execution strategy must be represented by an honest label.

LP:	$X \xrightarrow{\langle u, \alpha \rangle} W \xrightarrow{\langle v, \beta \rangle} Y$ $\langle u+v, \alpha\beta \rangle$	
$\mathbf{R_0}$**:**	$P? \xrightarrow[\langle w, \alpha' \rangle]{\langle w, \alpha\rho \rangle} X$	if $w < 0$, $\rho \in \{p, \neg p\}$
$\mathbf{R_3^*}$**:**	$P? \xrightarrow{\langle w, \alpha\beta \rangle} X \xleftarrow[\langle \max\{v,w\}, \alpha\beta\gamma' \rangle]{\langle v, \beta\gamma\rho \rangle} Y$	if $w \leq 0$, $\rho \in \{p, \neg p\}$

Pre-existing constraints (unshaded) are presumed to have honest, coherent, and satisfiable labels. New constraints (shaded) are only generated if their labels are satisfiable. For R_0, the literal ρ (either p or $\neg p$) does not appear in α or $L(X)$, and α' is obtained by removing any occurrence of ρ, as well as any children of ρ from α. (If ρ appeared in $L(X)$, then the WD_2 requirement that the network include a constraint, $(P? - X \leq -\epsilon, L(X))$, would, through an application of the LP rule, generate the unsatisfiable constraint, $(X - X \leq -\epsilon + w, \alpha)$ (i.e., $(0 < 0, \alpha)$), which would imply that the network was not DC, obviating the need for any further constraint generation.) For R_3^*, α, β and γ must not share any letters, ρ must not appear in $\alpha, \beta, \gamma, L(X)$ or $L(Y)$, and γ' is obtained by removing children of ρ from γ.

TABLE I: A subset of the CSTNU rules, restricted to CSTNs

LP:	$X \xrightarrow{\langle 3, pqr \rangle} W \xrightarrow{\langle 4, rs\neg t \rangle} Y$ $\langle 7, pqrs\neg t \rangle$	
$\mathbf{R_0}$**:**	$P? \xrightarrow[\langle -8, bcd \rangle]{\langle -8, abcdp \rangle} X,$	where $L(A?) = pr$
$\mathbf{R_3^*}$**:**	$P? \xrightarrow{\langle -5, ab \rangle} X \xleftarrow[\langle -3, abc \rangle]{\langle -3, bcdp \rangle} Y,$	where $L(D?) = p$
$\mathbf{R_3^*}$**:**	$P? \xrightarrow{\langle -5, ab \rangle} X \xleftarrow[\langle -5, abc \rangle]{\langle -8, bcdp \rangle} Y,$	where $L(D?) = p$

Unless stated otherwise, the label on each time-point is $\boxdot$.

TABLE II: Instances of the CSTN rules from Table I

Table I lists a set of three constraint-propagation rules for CSTNs that are restricted versions (with some modifications to ensure that they preserve label honesty) of constraint-propagation rules for CSTNUs from prior work [5], [6]; corresponding sample instances of these rules are given in Table II. For each rule, the labeled value generated by the rule is shaded. For example, the *Label Propagation* (LP) rule generates a labeled edge from X to Y when given pre-existing edges from X to W to Y. This rule is obtained by restricting the *Labeled No Case* rule for CSTNUs to CSTNs.[5] It is analogous to the propagation of unlabeled edges in an STN, except that the labels on the pre-existing edges are conjoined in the generated edge. (Recall that an expression such as $\alpha\beta$ represents the conjunction of the labels α and β.)

Next, the R_0 rule for CSTNs is a restriction of the R_0 rule for CSTNUs, modified slightly to ensure that the generated label is honest. R_0 is a *label modification* rule: the numerical weight of the edge does not change; but the label is made more general. The intuition behind this rule is that if the time-point X must be executed before the truth value of p can be known (i.e., if X must be executed before $P?$), then that constraint cannot be restricted to scenarios that depend on p and, thus, p—and any of p's children—can be removed from the label.

[5]The *Labeled No Case* rule for CSTNUs is a generalization of the *No Case* rule for STNUs introduced by Morris and Muscettola [11]. Since the descriptor "No Case" only provides intuitive guidance in the context of networks (e.g., STNUs and CSTNUs) that have contingent links, this paper chooses a more helpful descriptor for the CSTN rule: *Labeled Propagation.*

Finally, the R_3^* rule for CSTNs is a restriction of the R_3 rule for CSTNUs, but generalized to cover the case $v < w$. (Hence the asterisk.) It applies in cases where the execution of X and any *violation* of the generated constraint would both have to occur before the value of p is known. For example, in the first sample instance of R_3^* in Table II, if $X = 0$, then in scenarios consistent with abc, p cannot be observed before time 5 and any *violation* of the generated constraint, $(X - Y \leq -3, abc)$ (i.e., $(Y \geq 3, abc)$), would require executing Y before time 3, and hence before p could be observed. But that might lead to a subsequent violation of the pre-existing constraint, $(X - Y \leq -3, bcdp)$, should p and d both eventually happen to be $true$. Thus, the generated constraint must not be violated.

C. Soundness and Completeness

The notions of soundness and completeness are central to this paper. Therefore, they are defined carefully below, starting with the definition of what it means for an execution strategy to satisfy a labeled constraint, something that is only implicit in the definition of a *viable* execution strategy (cf. Defn. 11).

Definition 17. An execution strategy σ *satisfies* a labeled constraint $(Y - X \leq \delta, \ell)$ if for every scenario $s \in P^*$:

(1) s is inconsistent with ℓ; or
(2) $[\sigma(s)]_Y - [\sigma(s)]_X \leq \delta$.

Fact 1. *Let $\mathcal{S}$ be any CSTN, and σ any execution strategy for $\mathcal{S}$. Then σ is viable for $\mathcal{S}$ if and only if σ satisfies every labeled constraint in $\mathcal{S}$.*

Proof: Suppose σ is viable. Let s be any scenario. Since σ is viable, the schedule $\sigma(s)$ is a solution to the projection $scPrj(\mathcal{S}, s)$. Thus, $[\sigma(s)]_Y - [\sigma(s)]_X \leq \delta$ holds for every constraint $(Y - X \leq \delta)$ in that projection. Since the constraints in that projection correspond to the *labeled* constraints in $\mathcal{S}$ whose labels are consistent with s, and since the choice of s was arbitrary, it follows that σ satisfies every labeled constraint in $\mathcal{S}$. Furthermore, each of these steps is reversible. ■

Corollary 1. *A CSTN is dynamically consistent if and only if it has a dynamic execution strategy that satisfies all of its labeled constraints.*

As already mentioned, the DC-checking algorithm presented in this paper is based on the propagation of labeled constraints according to a particular set of constraint-propagation rules. The algorithm applies the rules in all possible combinations until either all possible propagations have been done or a particular kind of negative cycle has been found. (The algorithm is presented in detail in Section III-F.) If such a negative cycle is found, then no dynamic execution strategy can execute all of the time-points in the cycle while satisfying all of the constraints that determine the cycle; hence, the network is not dynamically consistent [1], [8]. A necessary condition for ensuring that the DC-checking algorithm properly distinguishes DC and non-DC networks is that the constraint-propagation rules must be sound, as follows.

Definition 18 (Sound Constraint-Propagation Rule)**.** A constraint-propagation rule for CSTNs is *sound* if whenever a *viable* and *dynamic* execution strategy σ satisfies the pre-existing constraints in any instance of that rule, then σ necessarily satisfies the corresponding constraint generated by that instance of the rule.

Theorem 1. *The constraint-propagation rules in Table I are sound.*

Proof: For the LP rule, suppose that σ is any execution strategy that satisfies the labeled constraints, $(W - X \leq u, \alpha)$ and $(Y - W \leq v, \beta)$. Let s be any scenario that is consistent with $\alpha\beta$. Then s is consistent with both α and β; therefore, $[\sigma(s)]_W - [\sigma(s)]_X \leq u$ and $[\sigma(s)]_Y - [\sigma(s)]_W \leq v$, which together imply that $[\sigma(s)]_Y - [\sigma(s)]_X \leq u + v$.

For the R_0 rule, suppose that σ is any viable and dynamic execution strategy that satisfies the labeled constraint, $(X - P? \leq w, \alpha p)$, where $w < 0$, and p does not appear in α (either positively or negatively), as in Table I. (The case $\rho = \neg p$ is handled similarly.) First, note that X and $P?$ must be distinct time-points since $0 = X - X \leq w < 0$ is unsatisfiable. Next, let s be any scenario that is consistent with α. Then, let s_1 and s_2 be the same as s, except that $s_1(p) = true$ and $s_2(p) = false$. Now, by construction, s_1 is consistent with αp; thus, $[\sigma(s_1)]_X - [\sigma(s_1)]_{P?} \leq w < 0$, whence $[\sigma(s_1)]_X < [\sigma(s_1)]_{P?}$. Next, consider the schedules, $\sigma(s_1)$ and $\sigma(s_2)$, each augmented with annotations indicating the truth values of observation time-points as they execute. Let $t^* \in \mathbb{R}$ be the first time at which these annotated schedules differ. By construction, $Hist(t^*, s_1, \sigma) = Hist(t^*, s_2, \sigma)$, whence the only way that the annotated schedules could differ at t^* is if some observation time-point is executed at t^* yielding different truth values in the two scenarios (cf. Defn. 15). Since s_1 and s_2 differ only with respect to p, it follows that $P?$ must be executed at time t^* in both scenarios (i.e., $[\sigma(s_1)]_{P?} = t^* = [\sigma(s_2)]_{P?}$). Given that $[\sigma(s_1)]_X < [\sigma(s_1)]_{P?} = t^*$, and the fact that the two schedules are identical before t^*, it follows that $[\sigma(s_2)]_X = [\sigma(s_1)]_X$ and therefore that $[\sigma(s_2)]_X - [\sigma(s_2)]_{P?} \leq w$. Since the scenario s equals one of s_1 and s_2, it follows that $[\sigma(s)]_X - [\sigma(s)]_{P?} \leq w$. Since s was any scenario consistent with α, it follows that σ satisfies the labeled constraint, $(X - P? \leq w, \alpha)$.

The above argument showed that p (or $\neg p$) could be removed from the label on a negative edge emanating from $P?$. A similar argument can be used to show that any child of p (or $\neg p$) can be removed from α, yielding a labeled constraint that σ continues to satisfy. The key point is that any child $Q?$ of $P?$ is constrained to occur after $P?$; therefore, by the time $Q?$ is executed, both X and $P?$ will have already been executed in a way that satisfies the inequality, $X - P? \leq w$. Proceeding in this way, removing one child literal at a time from α, ensures that σ must satisfy the labeled constraint, $(X - P? \leq w, \alpha')$, where α' is obtained by removing any children of p (or $\neg p$) from α.

For rule R_3^*, suppose that σ is a viable and dynamic execution strategy that satisfies the labeled constraints, $(X - P? \leq w, \alpha\beta)$

and $(X - Y \leq v, \beta\gamma p)$, where $w \leq 0$. (The case where $\rho = \neg p$ is handled similarly.) We first aim to show that σ must also satisfy the labeled constraint, $(X - Y \leq m, \alpha\beta\gamma)$, where $m = \max\{v, w\}$. Let s be any scenario that is consistent with $\alpha\beta\gamma$. As in the preceding proof, let s_1 and s_2 be the same as s except that $s_1(p) = \mathit{true}$ and $s_2(p) = \mathit{false}$. Now, since p does not appear in $\alpha\beta$ (cf. Table I) and $s \models \alpha\beta\gamma$, it follows that $s_1 \models \alpha\beta$. Therefore, since σ satisfies $(X - P? \leq w, \alpha\beta)$, it follows that $[\sigma(s_1)]_X - [\sigma(s_1)]_{P?} \leq w$ and, hence, that $[\sigma(s_1)]_X \leq [\sigma(s_1)]_{P?} + w \leq [\sigma(s_1)]_{P?}$. Similarly, it follows that $[\sigma(s_2)]_X - [\sigma(s_2)]_{P?} \leq w$ and $[\sigma(s_2)]_X \leq [\sigma(s_2)]_{P?}$.

Next, by construction, $s_1 \models \beta\gamma p$. In addition, σ satisfies $(X - Y \leq v, \beta\gamma p)$. Therefore, $[\sigma(s_1)]_X - [\sigma(s_1)]_Y \leq v \leq m$.

Next, as in the proof for rule R_0, let t^* be the first time at which the *annotated* schedules for $\sigma(s_1)$ and $\sigma(s_2)$ differ. As before, since s_1 and s_2 differ only with respect to p, it follows that $[\sigma(s_1)]_{P?} = t^* = [\sigma(s_2)]_{P?}$. And since both schedules execute X at or before $P?$, it follows that $[\sigma(s_1)]_X = [\sigma(s_2)]_X$. (If one schedule executes X at some time $t_x < t^*$, then both must execute X at t_x. The only other option is that both execute X at t^*.)

Finally, note that if Y is executed at some time $t_y < t^*$ in either schedule, then $\sigma(s_1)$ and $\sigma(s_2)$ both must execute Y at t_y, whence $[\sigma(s_2)]_X - [\sigma(s_2)]_Y = [\sigma(s_1)]_X - [\sigma(s_1)]_Y \leq m$. However, if $\sigma(s_2)$ executes Y at or after t^*, then $[\sigma(s_2)]_Y \geq t^* = [\sigma(s_2)]_{P?} \geq [\sigma(s_2)]_X - w \geq [\sigma(s_2)]_X - m$.

Since one of s_1 and s_2 is identical to s, it follows that $[\sigma(s)]_X - [\sigma(s)]_Y \leq m$. Since s was chosen arbitrarily such that $s \models \alpha\beta\gamma$, it follows that σ satisfies $(X - Y \leq m, \alpha\beta\gamma)$. All that remains is to show that any children of p can be removed from γ (yielding γ' in Table I), which can be done one at a time, as in the proof for rule R_0. ■

Note. It is straightforward to show that the LP, R_0 and R_3^* rules preserve label honesty and coherence.

In the field of temporal constraint networks it is common practice to classify a consistency checking algorithm as *sound* if its negative answer is always correct [7], [11], [12]. Similarly, *completeness* is typically defined in terms of its answers for non-DC networks. Therefore, this paper adopts the following definition of soundness and completeness for DC-checking algorithms for CSTNs.

Definition 19 (Soundness and Completeness for DC-checking algorithms)**.** A DC-checking algorithm for CSTNs is *sound* if its negative answer is always correct (i.e., whenever it says that a given CSTN is not DC, then the network is necessarily not DC). A DC-checking algorithm for CSTNs is *complete* if for any non-DC instance given as input, the algorithm always returns a negative answer. A DC-checking algorithm is *correct* if it is both sound and complete.

It is worth noting that if a DC-checking algorithm always halts with an answer, then the definition of completeness can be expressed using the contrapositive of the above definition: a DC-checking algorithm for CSTNs is *complete* if whenever it says that a given CSTN is DC, then that network is necessarily DC. That is the approach taken in Theorem 3 below.

For any DC-checking algorithm that always halts with an answer, the following table provides equivalent characterizations of the notions of soundness and completeness:

Sound:	Alg. says "No"	$\Longrightarrow$	CSTN is not DC
	CSTN is DC	$\Longrightarrow$	Alg. says "Yes"
Complete:	Alg. says "Yes"	$\Longrightarrow$	CSTN is DC
	CSTN is not DC	$\Longrightarrow$	Alg. says "No"

The DC-checking algorithm presented in this paper uses a sound set of constraint-propagation rules that generate constraints that any viable and dynamic execution strategy for any given CSTN must satisfy. Therefore, the algorithm is sound. Furthermore, the algorithm always terminates and, as will be seen, if it says that a given CSTN is DC, then the network is necessarily DC. Therefore, the algorithm is both sound and complete.

D. Extending literals and labels

To properly deal with the challenges raised by negative q-loops and negative q-stars, the labels on constraints must be more expressive: they must be able to represent that a constraint applies only in scenarios where the truth value of a given propositional letter is not yet known. Toward that end, *q-literals* and *q-labels* are defined below. For example, a q-labeled constraint such as $(X \geq 5, (?p)q)$ only applies in scenarios where p is unknown and q is true. For the purposes of this paper, it suffices to restrict the use of q-labels to lower-bound edges terminating at Z (i.e., the edges that play such an important role in the earliest-first execution strategy).

First, an alternative characterization of what it means for a strategy to satisfy a labeled constraint on a lower-bound edge is given. The following result essentially says that a viable and dynamic execution strategy σ satisfies the labeled constraint $(X \geq \delta, \ell)$ if and only if when X is executed in any scenario s, then either ℓ is already known to be false or the constraint holds: $[\sigma(s)]_X \geq \delta$.

Lemma 1. *Let σ be any viable and dynamic execution strategy for some CSTN $\mathcal{S}$; and let $(X \geq \delta, \ell)$ be any lower-bound constraint, where $\ell \in \mathcal{P}^*$. Let:*

$$\mathcal{P}_\ell^+ = \{p_i \in \mathcal{P} \mid \ell \models p_i \text{ and } X \not\equiv P_i?\}\text{; and}$$
$$\mathcal{P}_\ell^- = \{q_j \in \mathcal{P} \mid \ell \models \neg q_j \text{ and } X \not\equiv Q_j?\}.$$

Then σ satisfies $(X \geq \delta, \ell)$ if and only if for each scenario s, at least one of the following hold:

(1) $[\sigma(s)]_X \geq \delta$;
(2) $\bigvee_{p_i \in \mathcal{P}_\ell^+} ([\sigma(s)]_{P_i?} \leq [\sigma(s)]_X) \wedge (s(p_i) = \mathit{false})$; *or*
(3) $\bigvee_{q_j \in \mathcal{P}_\ell^-} ([\sigma(s)]_{Q_j?} \leq [\sigma(s)]_X) \wedge (s(q_j) = \mathit{true})$.

Note that condition (2) holds when some propositional letter p_i that appears positively in ℓ is already known to be false when X is executed. Similarly, condition (3) holds when some propositional letter q_j that appears negatively in ℓ is already known to be true when X is executed.

Proof: Suppose that σ is a viable and dynamic execution strategy that satisfies $(X \geq \delta, \ell)$ for some $\ell \in \mathcal{P}^*$, but that for

Fig. 4: Constraints that must be satisfied as long as q unknown

some scenario s, conditions (1), (2) and (3) are all false. In other words, *all* of the following hold:

(i) $[\sigma(s)]_X < \delta$;
(ii) $\bigwedge_{p_i \in \mathcal{P}_\ell^+}([\sigma(s)]_{P_i?} > [\sigma(s)]_X) \vee (s(p_i) = \mathit{true})$; and
(iii) $\bigwedge_{q_j \in \mathcal{P}_\ell^-}([\sigma(s)]_{Q_j?} > [\sigma(s)]_X) \vee (s(q_j) = \mathit{false})$.

Since σ satisfies the given constraint, but $[\sigma(s)]_X < \delta$, it follows that $s \not\models \ell$. Thus, there must be at least one propositional letter p that appears in both ℓ and s, but with opposite polarity. Let $\mathcal{P}_s$ be the set of propositional letters that appear in both ℓ and s, but with opposite polarity. Let s' be the same as s, except that for each $p \in \mathcal{P}_s$, $s'(p)$ is the opposite of $s(p)$. By its construction, $s' \models \ell$. Therefore, $[\sigma(s')]_X \geq \delta$. As in previous proofs, let $t^* \in \mathbb{R}$ be the first time at which the annotated schedules $\sigma(s)$ and $\sigma(s')$ differ. Since $\mathit{Hist}(t^*, s, \sigma) = \mathit{Hist}(t^*, s', \sigma)$, it follows that some observation time-point $P?$ must be executed at time t^* in both schedules, but such that $s(p) \neq s'(p)$. Now, if $p \in \mathcal{P}_s \cap \mathcal{P}_\ell^+$, then $s(p) = \mathit{false}$, which, by (ii) above, implies that $t^* = [\sigma(s)]_{P?} > [\sigma(s)]_X$, which implies that X must be executed at the same time in both schedules: $[\sigma(s)]_X = [\sigma(s')]_X$. Similarly, if $p \in \mathcal{P}_s \cap \mathcal{P}_\ell^-$, then $s(p) = \mathit{true}$, which, by (iii) above, implies that $t^* = [\sigma(s)]_{P?} > [\sigma(s)]_X$ and hence that $[\sigma(s)]_X = [\sigma(s')]_X$. The only other possibility is that X happens to be the observation time-point for $p \in \mathcal{P}_s$, and $[\sigma(s)]_X = t^* = [\sigma(s')]_X$. Thus, each case implies that X is executed at the same time by both schedules. But then $[\sigma(s')]_X \geq \delta$ implies that $[\sigma(s)]_X \geq \delta$, which is a contradiction. ■

The preceding lemma provides a convenient characterization of what it means for a viable and dynamic execution strategy σ to satisfy a labeled lower-bound constraint $(X \geq \delta, \ell)$. In essence, as long as the current partial scenario is consistent with the label ℓ (i.e., as long as ℓ is not yet known to be false), then X must be executed no earlier than δ. Now, if $\ell \in \mathcal{P}^*$, then ℓ can only be known to be false if some $p \in \mathcal{P}_\ell^+$ is already known to be false, or some $q \in \mathcal{P}_\ell^-$ is already known to be true. However, it turns out that it is also important to accommodate a more general kind of lower-bound constraint: for example, one that applies as long as the truth value of one or more propositional letters remains unknown (i.e., as long as the corresponding observation time-points have not yet been executed).

For example, consider the labeled constraints shown in Figure 4. As long as the truth value of q is not yet known (i.e., as long as $Q?$ has not yet been executed), both of these constraints must be satisfied. As a result, as long as q is unknown, Y must be at least 9 after Z. However, such a constraint is not representable using the labeled constraints seen so far.

Below, literals are extended to include expressions of the form $?p$, called *q-literals.* (The *q* stands for "question mark".) Intuitively, a constraint labeled by $?p$ must hold as long as the value of p is unknown. A *q-label* is a label that may contain one or more q-literals. For the purposes of this paper, q-labels are only needed for lower-bound edges terminating at Z.

Definition 20 (q-literals and q-labels)**.**

- A *q-literal* is a literal of the form $?p$, where $p \in \mathcal{P}$.
- A *q-label* is a conjunction of literals each of the form, $p, \neg p$ or $?p$, for some $p \in \mathcal{P}$.
- $\mathcal{Q}^*$ denotes the set of all q-labels.

For example, $p(?q)\neg r$ and $(?p)(?q)(?r)stu$ are both q-labels.

The semantics for satisfying a q-labeled lower-bound constraint is given in a form that mirrors the alternative characterization of the semantics for satisfying a labeled lower-bound constraint that was given in Lemma 1.

Definition 21 (Satisfying a Q-labeled Constraint)**.** Let σ be any viable and dynamic execution strategy for a CSTN $\mathcal{S}$; and let $(X \geq \delta, \ell)$ be a lower-bound constraint, where $\ell \in \mathcal{Q}^*$. Let:

$$\mathcal{P}_\ell^+ = \{p_i \in \mathcal{P} \mid p_i \in \ell \text{ and } X \not\equiv P_i?\};$$
$$\mathcal{P}_\ell^- = \{q_j \in \mathcal{P} \mid \neg q_j \in \ell \text{ and } X \not\equiv Q_j?\}; \text{ and}$$
$$\mathcal{P}_\ell^? = \{r_k \in \mathcal{P} \mid ?r_k \in \ell \text{ and } X \not\equiv R_k?\}.$$

Then σ satisfies $(X \geq \delta, \ell)$ if and only if for each scenario s, at least one of the following hold:

(1) $[\sigma(s)]_X \geq \delta$;
(2) $\bigvee_{p_i \in \mathcal{P}_\ell^+}([\sigma(s)]_{P_i?} \leq [\sigma(s)]_X) \wedge (s(p_i) = \mathit{false})$;
(3) $\bigvee_{q_j \in \mathcal{P}_\ell^-}([\sigma(s)]_{Q_j?} \leq [\sigma(s)]_X) \wedge (s(q_j) = \mathit{true})$; or
(4) $\bigvee_{r_k \in \mathcal{P}_\ell^?}[\sigma(s)]_{R_k?} \leq [\sigma(s)]_X$.

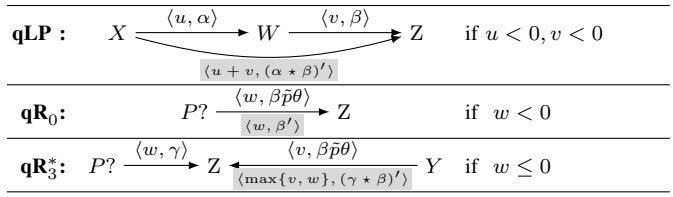
In other words, σ satisfies the q-labeled lower-bound constraint if $[\sigma(s)]_X \geq \delta$; some positive literal in ℓ is already known to be false; some negative literal in ℓ is already known to be true; or some q-literal in ℓ has already been observed (whether true or false).

To facilitate the presentation of the rules for propagating q-labeled constraints, the $\star$ operator is defined below. The intuition behind the $\star$ operator is illustrated by the following simple example. Suppose that C_1 is a constraint labeled by p, and C_2 is a constraint labeled by $\neg p$. Then C_1 must hold as long as p is either unknown or known to be true; C_2 must hold as long as p is either unknown or known to be false; and *both* C_1 and C_2 must hold as long as p is unknown—which is represented by the q-label, $p \star (\neg p) = ?p$.

Definition 22 (The $\star$ operator)**.** The commutative binary operator, $\star : \mathcal{Q}^* \times \mathcal{Q}^* \to \mathcal{Q}^*$, is defined in two steps. First, when given any combination of literals, $\lambda_1, \lambda_2 \in \{p, \neg p, ?p\}$, that together involve only one propositional letter, $\lambda_1 \star \lambda_2$ is defined by the following chart:

$\star$	p	$\neg p$	$?p$
p	p	$?p$	$?p$
$\neg p$	$?p$	$\neg p$	$?p$
$?p$	$?p$	$?p$	$?p$

Next, for any q-labels, $\ell_1, \ell_2 \in \mathcal{Q}^*$, the expression $\ell_1 \star \ell_2 \in \mathcal{Q}^*$ denotes the conjunction of literals obtained by applying the $\star$ operator in pairwise fashion to corresponding literals from ℓ_1 and ℓ_2, as follows.



In the above rules, $\alpha \in \mathcal{P}^*$; $\beta, \gamma, \theta \in \mathcal{Q}^*$; $\tilde{p} \in \{p, \neg p, ?p\}$; and expressions such as ℓ' represent the q-label obtained from ℓ by removing the children of any q-literals that appear in ℓ. In qR$_0$ and qR$_3^*$, β contains no children of $\tilde{p}$, and θ *only* contains children of $\tilde{p}$. In qR$_3^*$, γ does not contain $\tilde{p}$ or any of its children.

TABLE III: New constraint-propagation rules that accommodate q-labels on lower-bound edges involving Z

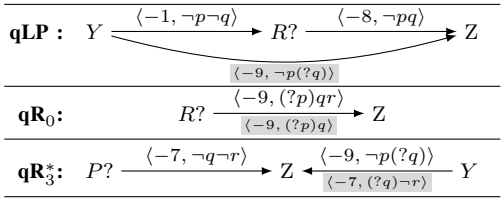


TABLE IV: Sample instances of the rules from Table III

- If $\lambda_1, \lambda_2 \in \{p, \neg p, ?p\}$ are literals in ℓ_1 and ℓ_2, respectively, that involve the same propositional letter p, then $\lambda_1 \star \lambda_2$ is contained in the conjunction, $\ell_1 \star \ell_2$.
- If p appears as λ_1 in ℓ_1, but p does not appear in ℓ_2, then λ_1 is contained in the conjunction, $\ell_1 \star \ell_2$.
- If p appears as λ_2 in ℓ_2, but p does not appear in ℓ_2, then λ_2 is contained in the conjunction, $\ell_1 \star \ell_2$.

For example:

$$\begin{aligned} p\neg q(?r)stu \star pqr\neg sv(?w) \\ &= (p \star p)(\neg q \star q)(?r \star r)(s \star \neg s)tuv(?w) \\ &= p(?q)(?r)(?s)tuv(?w) \end{aligned}$$

Table III provides a new set of constraint-propagation rules that accommodate q-labels on lower-bound constraints (i.e., on edges terminating at Z). The rules parallel those seen earlier in Table I. To highlight the parallels, the names of the new rules are the same as the old ones, except that they are prepended with the letter "q". Table IV illustrates the use of the new rules with examples drawn from the CSTN graph from Fig. 3.

For example, consider the instance of the qLP rule in Table IV. In any situation where p is *false* and q is not yet known, both pre-existing constraints must be satisfied and, hence, the generated constraint must also be satisfied. In particular, if Y must be at least 1 after $R?$, and $R?$ must be at least 8 after Z, then Y must be at least 9 after Z. Note that this rule would not apply if the numerical value on the edge from Y to $R?$ was $+1$ because, in that case, for example, Y and $Q?$ could both be executed at, say, time 4, after which the observed value of q, whether *true* or *false*, would make one of the pre-existing constraints/edges irrelevant. In particular, if q was observed to be *true*, then $R?$ could be executed safely at 8, but if q was *false*, then $R?$ could be executed instead at 5. Such reactive strategies are not available when both pre-existing edges have negative values.

The instance of qR$_0$ in Table IV occurs midway through the propagation of constraints for the graph in Fig. 3. The "pre-existing" edge from $R?$ to Z is obtained from a prior application of the qLP rule to the edges from $R?$ to $Q?$ to Z. The qR$_0$ rule merely removes r from the labeled edge from $R?$ to Z since the value of r cannot be known before $R?$ executes.

Finally, the instance of qR$_3^*$ in Table IV uses the previously generated edge from Y to Z, and the pre-existing edge from $P?$ to Z. This instance models that in any situation where q is unknown and r is *false*, $P?$ cannot occur before time 7 (i.e., p cannot be known before 7), thus Y cannot occur before 7.

Before proving that the constraint-propagation rules in Table III are sound, it is helpful to prove the following lemma.

Lemma 2. *If σ is a viable and dynamic execution strategy that satisfies the labeled q-constraint, $(X \geq \delta, \ell)$, then σ must also satisfy $(X \geq \delta, \ell')$, where ℓ' is obtained by removing the children of any q-literals from ℓ*

Proof: Suppose that $\ell = \alpha(?p)\tilde{q}$, where $q \in \{q, \neg q, ?q\}$, and $Q?$ is a child of $P?$. Let s be any scenario. Suppose that conditions (2)-(4) from Defn. 21 are false for $(X \geq \delta, \alpha(?p))$. Then the analogous conditions for $(X \geq \delta, \alpha(?p)\tilde{q})$ must also be false unless one of the following holds:

(q.2) $(\tilde{q} \equiv q) \ \wedge ([\sigma(s)]_{Q?} \leq [\sigma(s)]_X) \wedge (s(q) = \mathit{false})$
(q.3) $(\tilde{q} \equiv \neg q) \wedge ([\sigma(s)]_{Q?} \leq [\sigma(s)]_X) \wedge (s(q) = \mathit{true})$
(q.4) $(\tilde{q} \equiv ?q) \ \wedge ([\sigma(s)]_{Q?} \leq [\sigma(s)]_X)$

Note that, if any of these hold, $[\sigma(s)]_{Q?} \leq [\sigma(s)]_X$. On the other hand, condition (4) being false for $(X \geq \delta, \alpha(?p))$ also implies that $[\sigma(s)]_{P?} > [\sigma(s)]_X$. But that implies that $[\sigma(s)]_{Q?} < [\sigma(s)]_{P?}$, which contradicts that σ, being viable, must execute $P?$ before any of its children. Thus, none of (q.2), (q.3) or (q.4) can hold. But then conditions (2)-(4) must be false for $(X \geq \delta, \alpha(?p)\tilde{q})$, which implies that condition (1) must hold (i.e., $[\sigma(s)]_X \geq \delta$), whence σ satisfies $(X \geq \delta, \alpha(?p))$. In the same way, any other children of any q-literals in ℓ can be removed, yielding a constraint that σ must satisfy. ■

Theorem 2. *The propagation rules for q-labeled constraints in Table III are sound.*

Proof: For the qLP rule, suppose σ is a viable and dynamic execution strategy that satisfies the constraints $(W - X \leq u, \alpha)$ and $(W \geq -v, \beta)$, where $u < 0$ and $v < 0$. First, consider a new constraint, $(X \geq -u - v, \ell)$, where $\ell = \alpha \star \beta$. (Note that $-u$ and $-v$ are both positive.) Let s be any scenario. Suppose that conditions (2)-(4) in Defn. 21 are all false for $(X \geq -u - v, \ell)$. In other words:

(ii) $\bigwedge_{p_i \in \mathcal{P}_\ell^+} ([\sigma(s)]_{P_i?} > [\sigma(s)]_X) \vee (s(p_i) = \mathit{true})$
(iii) $\bigwedge_{q_j \in \mathcal{P}_\ell^-} ([\sigma(s)]_{Q_j?} > [\sigma(s)]_X) \vee (s(q_j) = \mathit{false})$
(iv) $\bigwedge_{r_k \in \mathcal{P}_\ell^?} ([\sigma(s)]_{R_k?} > [\sigma(s)]_X)$

Let s' be the same as s except that for each p that appears positively in α, $s'(p) = \mathit{true}$, and for each p that appears negatively in α, $s'(p) = \mathit{false}$. Thus, $s' \models \alpha$. Since σ satisfies $(W - X \leq u, \alpha)$, it follows that $[\sigma(s')]_W - [\sigma(s')]_X \leq u < 0$, whence $[\sigma(s')]_W < [\sigma(s')]_X$. In addition, since the only

differences between s and s' occur after $[\sigma(s')]_X$, it follows that $[\sigma(s)]_X = [\sigma(s')]_X$ and $[\sigma(s)]_W = [\sigma(s)]_W$. As a result, $[\sigma(s)]_W - [\sigma(s)]_X \leq u$.

Next, suppose that p is any letter that appears positively in β. Then p appears as either p or $?p$ in $\alpha \star \beta$. In the first case, $([\sigma(s)]_{P?} > [\sigma(s)]_X > [\sigma(s)]_W) \vee (s(p) = true)$, by (ii) above. In the second case, $[\sigma(s)]_{P?} > [\sigma(s)]_X > [\sigma(s)]_W$, by (iv) above. Therefore, $([\sigma(s)]_{P?} > [\sigma(s)]_W) \vee (s(p) = true)$ holds in either case. Similarly, for any p that appears negatively in β, $([\sigma(s)]_{P?} > [\sigma(s)]_W) \vee (s(p) = false)$ holds. Finally, if p is any letter that appears as $?p$ in β, then $?p$ is also in $\alpha \star \beta$, whence $[\sigma(s)]_{P?} > [\sigma(s)]_W$ holds. Since these conditions all hold, and since σ satisfies $(W \geq -v, \beta)$, it follows that $[\sigma(s)]_W \geq -v$. But then

$$[\sigma(s)]_X = ([\sigma(s)]_X - [\sigma(s)]_W) + [\sigma(s)]_W \geq -u - v.$$

Thus, since s was arbitary, σ satisfies $(X \geq -u - v, \alpha \star \beta)$. And, by Lemma 2, σ must satisfy the same constraint where the children of any q-literals in $\alpha \star \beta$ have been removed.

For the qR_0 rule, suppose that σ satisfies the constraint, $(P? \geq -w, \beta\tilde{p}\theta)$, where $-w > 0$, $\tilde{p} \in \{p, \neg p, ?p\}$, β contains no children of $\tilde{p}$, and θ *only* contains children of $\tilde{p}$. Consider the constraint, $(P? \geq -w, \beta)$. Let s be any scenario. In Defn. 21, where $P? \equiv X$, note that p does not belong to any of the sets $\mathcal{P}^+_\ell, \mathcal{P}^-_\ell$ or $\mathcal{P}^?_\ell$. Furthermore, if $Q?$ is any child of $P?$, then the viability of σ ensures that $[\sigma(s)]_{Q?} > [\sigma(s)]_{P?}$. As a consequence, conditions (1)-(4) for satisfying $(P? \geq -w, \beta\tilde{p}\theta)$ are equivalent to conditions (1)-(4) for satisfying $(P? \geq -w, \beta)$. Thus, σ satisfies the latter constraint. Given Lemma 2, it follows that σ satisfies $(P? \geq -w, \beta')$, where β' is obtained by removing the children of any q-literals from β.

For the qR^*_3 rule, suppose that σ satisfies the constraints, $(P? \geq -w, \gamma)$ and $(Y \geq -v, \beta\tilde{p}\theta)$, where $-w$ and $-v$ are both positive, $\tilde{p} \in \{p, \neg p, ?p\}$, γ and β do not contain any children of $\tilde{p}$, and θ *only* contains children of $\tilde{p}$. Consider the new constraint, $(Y \geq -m, \gamma \star \beta)$, where $m = \max\{w, v\}$ (i.e., $-m = \min\{-w, -v\}$). Let s be any scenario.

Suppose that conditions (2)-(4) of Defn. 21 for the constraint, $(Y \geq -m, \gamma \star \beta)$, are all false, where $\ell = \gamma \star \beta$. In other words:

(i) $\bigwedge_{p_i \in \mathcal{P}^+_\ell} ([\sigma(s)]_{P_i?} > [\sigma(s)]_Y) \vee (s(p_i) = true)$;
(ii) $\bigwedge_{q_j \in \mathcal{P}^-_\ell} ([\sigma(s)]_{Q_j?} > [\sigma(s)]_Y) \vee (s(q_j) = false)$; and
(iii) $\bigwedge_{r_k \in \mathcal{P}^?_\ell} [\sigma(s)]_{R_k?} > [\sigma(s)]_Y$.

Next, consider the letters appearing in γ. First, any p_i that appears positively in γ, must appear in $\alpha \star \beta$ as either p_i or $?p_i$ (i.e., $p_i \in \mathcal{P}^+_\ell$ or $p_i \in \mathcal{P}^?_\ell$). Thus, by conditions (i) and (iii) above, it follows that $([\sigma(s)]_{P_i?} > [\sigma(s)]_{P?}) \vee (s(p_i) = true)$. Similarly, any q_j that appears negatively in γ, must appear in $\alpha \star \beta$ as either $\neg q_j$ or $?q$. Thus, by conditions (ii) and (iii) above, it follows that $([\sigma(s)]_{Q_j?} > [\sigma(s)]_{P?}) \vee (s(q_j) = false)$. And, any r_k that appears as $?r_k$ in γ, can only appear as $?r_k$ in $\gamma \star \beta$. Thus, by condition (iii) above, it follows that $[\sigma(s)]_{R_k?} > [\sigma(s)]_{P?}$. As a result, since σ satisfies $(P? \geq -w)$, it follows that $[\sigma(s)]_{P?} \geq -w$.

Next, recall that σ satisfies the constraint $(Y \geq -v, \beta\tilde{p}\theta)$. As above, suppose that conditions (2)-(4) from Defn. 21 are false for this constraint. Then, as argued earlier: for any p_i appearing positively in β, $([\sigma(s)]_{P_i?} > [\sigma(s)]_Y) \vee (s(p_i) = true)$ holds; for any q_j appearing negatively in β, $([\sigma(s)]_{Q_j?} > [\sigma(s)]_Y) \vee (s(q_j) = false)$ holds; and for any r_k appearing as $?r_k$ in β, $[\sigma(s)]_{R_k?} > [\sigma(s)]_Y$ holds. Now, suppose that $[\sigma(s)]_{P?} > [\sigma(s)]_Y$. Since σ is viable, it follows that $[\sigma(s)]_{Q?} > [\sigma(s)]_Y$ for any $Q?$ that is a child of $P?$. But in that case, conditions (2)-(4) from Defn. 21 for the constraint $(Y \geq -v, \beta\tilde{p}\theta)$ are all false. Since σ satisfies this constraint, it follows that condition (1) must be true, whence $[\sigma(s)]_Y \geq -v \geq -m$. On the other hand, if $[\sigma(s)]_{P?} \leq [\sigma(s)]_Y$, then $[\sigma(s)]_Y \geq [\sigma(s)]_{P?} \geq -w \geq -m$. Thus, in either case, $[\sigma(s)]_Y \geq -m$. Since s was arbitrary, it follows that σ satisfies $(Y \geq -m, \gamma \star \beta)$. And, given Lemma 2, σ must also satisfy $(Y \geq -m, (\gamma \star \beta)')$, whose label is obtained from $\gamma \star \beta$ by removing the children of any q-literals in $\gamma \star \beta$. ■

E. Negative Q-Stars and The Spreading Lemma

Prior to any constraint propagation, each time-point in Fig. 3 has one or more lower bounds that depend on the possible scenarios that may play out over time. For example, $R?$ has a lower bound of 2 in scenarios where p is *false* and q is *true*, and a lower bound of 0 in all other scenarios. However, during constraint propagation, the qLP rule typically generates stronger lower bounds. For example, the qLP rule generates a lower bound of 9 for Y in scenarios where p is *false* and q is unknown, as seen previously in Table IV.

At any given point during execution, the current partial scenario (CPS) is a label representing the observations that have occurred so far.

Definition 23 (Current Partial Scenario)**.** Let s be any scenario and $\mathcal{T}_o \subseteq \mathcal{OT}$ any subset of observation time-points. Then the *current partial scenario* (CPS) for that set of observation time-points in that scenario is notated as:

$$\text{CPS}(\mathcal{T}_o, s) = \begin{cases} \bigwedge\{p \mid P? \in \mathcal{T}_o \text{ and } s(p) = true\} \\ \wedge \\ \bigwedge\{\neg p \mid P? \in \mathcal{T}_o \text{ and } s(p) = false\} \end{cases}$$

Although a CPS bears some resemblance to a history (cf. Defn. 12), it is different in the following respects. First, the CPS is a function of a set of observation time-points, whereas the history is a function of an execution strategy and a time. Second, given the assumption about instantaneous reactivity made by this paper, several observation time-points might be executed at the same time, leading to instantaneous updates of the CPS. For example, executing $P?$ at time 7 might yield $p = true$, in which case an agent might wish to instantaneously react by executing $Q?$ at time 7 which might yield $q = false$. Thus, the CPS might first be $\boxdot$, then p, then $p\neg q$—all at time 7.

Since a dynamic strategy is able to react to observations, and certain observations can make a variety of constraints inapplicable, it is important to specify whether a q-labeled constraint is applicable, given the current partial scenario. For example, consider the constraint, $(X \geq 5, qr(?s)t)$. If the CPS

is, say, pq, then s is currently unknown, q is known to be true, and r and t *might* end up being true. Therefore, that constraint must not be violated (i.e., X must not be executed before 5). However, if the CPS is $qstu$, then that constraint is forever afterward inapplicable because the value of s is known. More generally, an expression of the form $appl(\ell', \ell)$ is used to represent that a constraint labeled by $\ell' \in \mathcal{Q}^*$ applies in the current partial scenario $\ell \in \mathcal{P}^*$ (i.e., has not been made irrelevant by the information contained in the CPS).

Definition 24 (Applicable Constraint)**.** Let $(X \geq \delta, \ell' \in \mathcal{Q}^*)$ be any q-labeled lower-bound constraint. That constraint is said to be *applicable* (or *relevant*) with respect to the current partial schedule ℓ—notated $appl(\ell', \ell)$—if for each propositional letter p that appears in both ℓ' and ℓ, p appears *identically* in both (i.e., as p in both, or as $\neg p$ in both).

Lemma 3. *Suppose that σ is a viable and dynamic execution strategy. Then σ satisfies the q-labeled constraint $(X \geq \delta, \ell')$ if and only if for each scenario s, $[\sigma(s)]_X < \delta$ implies that $appl(\ell', \ell)$ is false, where ℓ is the current partial scenario at the point where X is executed.*

Proof: If σ satisfies the constraint, then $[\sigma(s)]_X \geq \delta$ or one of the disjuncts in (2), (3) or (4) from Defn. 21 must hold. A disjunct from clause (2) implies that some p that appears positively in ℓ' is known to be false at or before the time X is executed. But that implies that $\neg p$ is in the CPS, while p is in ℓ', whence $appl(\ell', \ell)$ does not hold. Similar remarks apply to clause (3) and some p appearing negatively in ℓ'. Finally, a disjunct from clause (4) implies that some $?p$ appears in ℓ', but its observation time-point $P?$ has already been executed at or before X, implying that p or $\neg p$ is in the CPS, while $?p$ is in ℓ', again implying that $appl(\ell', \ell)$ does not hold. ■

Note. It suffices to restrict attention to current partial scenarios represented by honest labels, since any viable and dynamic execution strategy cannot execute a child of $P?$ before it executes $P?$.

For any CPS ℓ, and time-point X, the effective lower bound for X relative to ℓ is the *maximum* of its lower bounds among those having labels applicable to ℓ (i.e., having a label ℓ' such that $appl(\ell', \ell)$ holds). For example, for the network in Fig. 3, the ELB for $P?$ in the initial partial scenario is 7. Since that is the smallest ELB among all of the time-points other than Z, the earliest-first strategy will execute $P?$ first, at time 7, as shown in Algorithm 1. Since the execution of $P?$ generates a truth value for p, the CPS would then be updated to reflect that observation. Now, any constraint whose label ℓ' is inapplicable with the updated CPS can henceforth be ignored. As a result, the ELB values for the remaining unexecuted time-points may decrease in response to a smaller number of applicable constraints.

Rules qR_0 and qR_3^* work together to "spread" the *weakest* ELB for any given partial scenario to *every* unexecuted time-point, attaching the most general label possible: that of the CPS. For example, in Fig. 3, the weakest ELB for the initial partial scenario is 7. Rules qR_0 and qR_3^* spread that value to every other non-Z time-point, with the result that every edge that terminates at Z is annotated with the labeled value, $\langle -7, \boxdot \rangle$. More generally, the *spreading lemma,* presented below, ensures that this kind of spreading of effective lower bounds occurs relative to every possible current partial scenario. This property is invaluable in proving that the new DC-checking algorithm is complete.

Lemma 4 (Spreading Lemma)**.** *Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, \mathcal{P} \rangle$ be a well-defined CSTN whose constraint set $\mathcal{C}$ is closed under rules qR_0 and qR_3^*.[6] Let ℓ be an honest label representing the current partial scenario. Let $\mathcal{T}_o = \{P? \mid p \text{ appears in } \ell\}$ be the corresponding set of already-executed observation time-points. Let $\mathcal{T}_\ell = \{X \in \mathcal{T} \mid \ell \models L(X)\}$ be the set of time-points whose labels are entailed by ℓ. (The honesty of ℓ ensures that $\mathcal{T}_o \subseteq \mathcal{T}_\ell$.) Let $\mathcal{T}_x$ be any set of time-points such that $\mathcal{T}_o \subseteq \mathcal{T}_x \subset \mathcal{T}_\ell$. ($\mathcal{T}_x$ is the set of already-executed time-points.) And let $\mathcal{T}_u = \mathcal{T}_\ell - \mathcal{T}_x$ be the set of unexecuted time-points whose labels are entailed by ℓ. Then for each $X \in \mathcal{T}_u$, let:*

$$ELB(X, \ell) = \max\{\delta \mid \exists \ell' \in \mathcal{Q}^* : (X \geq \delta, \ell') \in \mathcal{C} \text{ and } appl(\ell', \ell)\};$$

and let $\Lambda(\ell) = \min\{ELB(X, \ell) \mid X \in \mathcal{T}_u\}$.

Then for each $X \in \mathcal{T}_u$, the constraint, $(X \geq \Lambda(\ell), \ell)$, is entailed by constraint(s) in $\mathcal{C}$.

Proof: By construction, if an observation time-point is in $\mathcal{T}_u$, then it has not yet been executed, but its label is entailed by ℓ; thus, it could be selected for execution next. Let $\mathcal{P}_u = \{p \mid P? \in \mathcal{T}_u\}$ be the set of propositional letters whose corresponding observation time-points are in $\mathcal{T}_u$. By construction, for each $p \in \mathcal{P}_u$, $\ell \models L(P?)$, but p does not appear in ℓ; and there must be a constraint, $(P? \geq \lambda_p, \ell_p)$ in $\mathcal{C}$, for some $\lambda_p \geq \Lambda(\ell)$, and $\ell_p \in \mathcal{Q}^*$ such that $appl(\ell_p, \ell)$. Now, if p appears in ℓ_p, then by Rule qR_0, it can be removed from ℓ_p and, since $\mathcal{C}$ is closed under qR_0 and qR_3^*, the resulting constraint must be entailed by constraints in $\mathcal{C}$. Thus, assume that p does not appear in ℓ_p. Next, let r be any other letter in $\mathcal{P}_u$. Again, there must be a constraint, $(R? \geq \lambda_r, \ell_r)$, for some $\lambda_r \geq \Lambda(\ell)$, and $\ell_r \in \mathcal{Q}^*$ such that $appl(\ell_r, \ell)$. Now, if p appears in ℓ_r, then it can be removed (along with any of its children) using Rule qR_3^*, resulting in the constraint, $(R? \geq \lambda_{pr}, \ell_{pr})$, where $\lambda_{pr} = \min\{\lambda_p, \lambda_r\} \geq \Lambda(\ell)$, and ℓ_{pr} is obtained by removing p, any children of p, and any children of any q-literals from $\ell_p \star \ell_r$. And since $appl(\ell_p, \ell)$ and $appl(\ell_r, \ell)$ both hold, so does $appl(\ell_{pr}, \ell)$. As before, since $\mathcal{C}$ is closed under rules qR_0 and qR_3^*, this constraint must be entailed by constraints in $\mathcal{C}$. Thus, without loss of generality, assume that p does not appear in ℓ_r. Continuing in this way, every observation time-point $T? \in \mathcal{T}_u$ must have a corresponding lower-bound constraint with lower bound $\lambda_t \geq \Lambda(\ell)$, whose propositional label ℓ_t does not include p. Once p has been removed from all such labels, it follows that r can similarly be removed from all such labels, and so on,

[6] In other words, the application of qR_0 and qR_3^* to constraints in $\mathcal{C}$ can only yield a constraint, χ, that is entailed by constraints already in $\mathcal{C}$—in the sense that any viable and dynamic execution strategy that satisfies the constraints in $\mathcal{C}$ necessarily satisfies the constraint χ.

Algorithm 2: CSTN_DC_Check($\mathcal{S}$)

Input: $\mathcal{S} = \langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P \rangle$, a well-defined CSTN.
$h =$ horizon value; // e.g., sum of all absolute edge weights
foreach $(X \in \mathcal{T})$ **do**
  Insert $(\mathrm{Z} - X \leq 0, \boxdot)$ and $(X - \mathrm{Z} \leq h, \boxdot)$;
do
  foreach $(X \in \mathcal{T})$ **do**
    if *$(X \in \mathcal{OT})$* **then**
      Apply $\mathrm{qR}_0(X, \mathrm{Z})$;
    foreach $(Y \in \mathcal{T})$ **do**
      Apply $\mathrm{qLP}(X, Y, \mathrm{Z})$;
      if *$(X \in \mathcal{OT})$* **then**
        Apply $\mathrm{qR}_3^*(X, \mathrm{Z}, Y)$;
        Apply $\mathrm{R}_0(X, Y)$;
      foreach $(W \in \mathcal{T})$ **do**
        Apply $\mathrm{LP}(X, Y, W)$;
        if *$(X \in \mathcal{OT})$* **then**
          Apply $\mathrm{R}_3^*(X, Y, W)$;
  if *(a negative one-edge loop with consistent label in P^* has been generated)* **then**
    return $\mathcal{S}$ *is* not *DC*
while *(any constraint has been added/updated)*;
return $\mathcal{S}$ *is DC*;

until each observation time-point $T? \in \mathcal{T}_u$ is seen to have a lower-bound constraint, $(T? \geq \lambda_t, \ell_t)$, where $\lambda_t \geq \Lambda(\ell)$ and ℓ_t contains no propositional letters from $\mathcal{P}_u$. Furthermore, since the rules preserve honesty, all such labels cannot have any children of any letters in $\mathcal{P}_u$. As a result, each label ℓ_t can only have letters from ℓ and, since $appl(\ell_t, \ell)$ holds, it follows that $\ell \models \ell_t$.[7] Finally, qR_3^* can then be used to similarly process the labels from lower-bound constraints on *all* time-points in $\mathcal{T}_u$, effectively removing from those labels any occurrences of letters in $\mathcal{P}_u$. Thus, for any $X \in \mathcal{T}_u$, $(X \geq \Lambda(\ell), \ell)$ must be entailed by constraint(s) in $\mathcal{C}$. ■

F. A Complete DC-Checking Algorithm

Algorithm 2 gives pseudo-code for our propagation-based DC-checking algorithm for CSTNs. Its core involves the application of the rules LP, R_0, R_3^*, qLP , qR_0 and qR_3^* (cf. Tables I and III). The triply nested *for* loops apply the rules to each pair or triple of time-points, as appropriate. (The pseudo-code is complicated by the fact that different rules apply to different numbers and types of input. As a result, the time-point names in expressions such as "Apply $\mathrm{qLP}(X, Y, Z)$" correspond with, but are not necessarily the same as those that appear in Tables I and III.) A constraint's labeled value is updated only if the new value is stronger than the current value with the same label. The algorithm ends when no stronger constraints can be generated or a negative one-edge loop with a consistent label in $\mathcal{P}^*$ has been generated. In the first case, the network is DC, in the other it is not.

Termination: The DC-checking algorithm employs two mechanisms to ensure termination. First, when using the qLP rule from Table III to update lower-bound edges terminating at Z, it keeps track of all of the time-points encountered so far in generating that update. Because this rule only propagates along negative edges, any time-point being visited more than once immediately signals the presence of a negative q-loop having all-negative edges. The algorithm avoids endlessly cycling through such a loop by immediately generating a labeled value of the form, $\langle -\infty, \ell' \rangle$, where ℓ' includes the conjunction of all labels along that loop. If ℓ' is a consistent label in $\mathcal{P}^*$, then the negative q-loop is unresolvable, and the network is instantly recognized to be non-DC; otherwise, this labeled value is propagated like any other. Such propagations will generate lower-bound labeled values of $\langle -\infty, \ell' \rangle$ for each time-point in the negative q-loop. For example, in Fig. 3, the negative q-loop from Y to $R?$ to $Q?$ leads to labeled values of $\langle -\infty, (?p)(?q)r \rangle$ on the lower-bound edges for $Y, Q?$ and $R?$. However, rules qR_0 and qR_3^* subsequently remove the literals, $?q$ and r, producing the occurrences of the labeled value, $\langle -\infty, ?p \rangle$, in the figure.

Second, because negative q-loops having one or more *non-negative* edges may lead to a large, but not necessarily infinite number of cycles, it would be unsound to immediately generate labeled values of the form, $\langle -\infty, \ell' \rangle$, for such loops. To accommodate the potentially bounded cycling of such loops, while terminating cases of infinite cycling, the algorithm inserts a universal upper bound h, called a *horizon.* For complete generality, h should be treated as a constant whose value is unknown, but arbitrarily large. In practice, this is not necessary since the networks in most applications already include finite horizons. Any time-point whose lower bound is updated to a value greater than h will lead to a negative edge from Z to Z which, being unsatisfiable, signals that the network is non-DC.

Theorem 3. *The DC-checking algorithm for CSTNs given in Algorithm 2 is* complete *(i.e., if the algorithm says that a given CSTN is DC, then the network is DC).*

Proof: Let $\mathcal{S}$ be a CSTN that the DC-checking algorithm says is DC. Let σ be the earliest-first execution strategy. The goal is to demonstrate that σ is a viable, dynamic execution strategy. Now, σ is a *dynamic* strategy since each of its execution decisions depends only on the history of *past-or-present* observations represented by the current partial scenario. Let s be any scenario, and $\mathcal{S}(s)$ the corresponding projection of $\mathcal{S}$ onto s. It remains to show that the schedule $\sigma(s)$ is a solution to the STN $\mathcal{S}(s)$. It is important to stress that each edge in $\mathcal{S}(s)$ has a label that is consistent with s, and was *originally present* in the CSTN $\mathcal{S}$ before any constraint propagation.

Now, suppose that $\mathcal{S}(s)$ is an inconsistent STN. Then it must contain a negative loop. But the corresponding negative loop in $\mathcal{S}$ would, by repeated use of the LP rule during constraint

[7] If ℓ_t contained a letter p_1 that was not in ℓ or $\mathcal{P}_u$, and was not a child of any such letter, then p_1 would have to be the child of a letter p_2 that was not in ℓ or $\mathcal{P}_u$, or a child of any such letter, and so on, leading to a circle of child nodes, which cannot happen in a well-defined CSTN since the label on each child must properly entail the label on its parent.

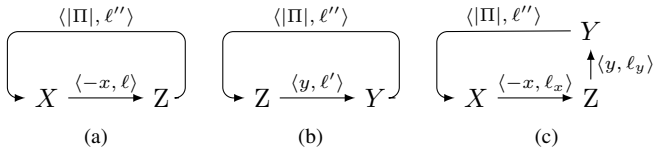


Fig. 5: Paths discussed in the proof of Theorem 3

propagation, yield a *single-edge* negative loop that would cause the DC-checking algorithm to say that the CSTN $\mathcal{S}$ was not DC. Thus, $\mathcal{S}(s)$ must be a consistent STN.

Next, suppose that the schedule $\sigma(s)$ is *not* a solution for $\mathcal{S}(s)$. For each time-point X, let $x = [\sigma(s)]_X$ be the value assigned to X by $\sigma(s)$. The corresponding execution constraints are $\mathrm{Z} - X \leq -x$ and $X - \mathrm{Z} \leq x$ (i.e., $X = x$). Since $\sigma(s)$ is not a solution for $\mathcal{S}(s)$, inserting these execution constraints into $\mathcal{S}(s)$ must yield a negative loop [8]. And, without loss of generality, there must be a negative loop having only one occurrence of Z. Let $\mathcal{L}$ be any such loop.

Case 1. In this case, as illustrated in Fig. 5a, $\mathcal{L}$ consists of a lower-bound edge from X to Z, followed by a path Π from Z back to X, where: (1) the label ℓ on the lower-bound edge represents the current partial scenario when X was executed; (2) the lower bound on that edge satisfies: $x = \Lambda(\ell) = ELB(X, \ell)$; (3) the edges in Π are original edges from $\mathcal{S}(s)$ having labels consistent with s; (4) ℓ'' is the conjunction of those labels; and (5) $|\Pi| < x$. The spreading lemma ensures that the constraint, $(\mathrm{Z} - X \leq -x, \ell)$, is entailed by constraints in the (propagated) CSTN. But, then, since ℓ and ℓ'' are consistent with s, constraint propagation would have yielded a negative loop consisting of the edge, $(\mathrm{Z} - \mathrm{Z} \leq |\Pi| - x, \ell \wedge \ell'')$, whose label is consistent with s. But then the algorithm would have reported that $\mathcal{S}$ was not DC, contradicting the main premise.

Case 2. In this case, as illustrated in Fig. 5b, $\mathcal{L}$ consists of an upper-bound edge from Z to Y, followed by a path Π from Y back to Z, where: (1) the label ℓ' on the upper-bound edge represents the current partial scenario when Y was executed; (2) the upper bound on that edge is $y = \Lambda(\ell') = ELB(Y, \ell')$; (3) the edges in Π are original edges from $\mathcal{S}(s)$ whose labels are consistent with s; (4) ℓ'' is the conjunction of those labels; and (5) $|\Pi| < -y$. In this case, repeated application of the LP rule would have yielded the edge, $(\mathrm{Z} - Y \leq |\Pi|, \ell'')$, where ℓ'' is consistent with s and, hence, also with ℓ'. But then, $-|\Pi| \leq ELB(Y, \ell') = y < -|\Pi|$, a contradiction.

Case 3. This case, illustrated in Fig. 5c, considers the possibility of a negative loop formed by a lower-bound constraint from the execution of some X together with an upper-bound constraint from the execution of some other Y. Here, $|\Pi| - x + y < 0$ and $\ell_x \wedge \ell_y \wedge \ell''$ is consistent with s. First, suppose that $x \leq y$ (i.e., X executed no later than Y). By Case 1, the lower-bound constraint for X is already entailed by constraints in $\mathcal{C}$. Thus, repeated application of the LP rule to the path consisting of Π followed by the lower-bound edge for X would have yielded the edge, $(\mathrm{Z} - Y \leq |\Pi| - x, \ell_x \wedge \ell'')$. By construction, its label is consistent with ℓ_y, implying that $-|\Pi| + x$ is a relevant lower bound for X. Hence, $y = ELB(Y, \ell_y) \geq -|\Pi| + x > y$, a contradiction.

Next, suppose that Y executes before X. Then by Case 2, the upper-bound edge for Y cannot introduce a negative loop. Furthermore, since that edge emanates from Z, its insertion cannot affect the lengths of any edges terminating at Z, and thus cannot affect the value of $ELB(X, \ell_x)$. Therefore, the Case 1 conclusion that the lower-bound edge for X is already entailed by constraints in $\mathcal{C}$ stands. Thus, its subsequent insertion cannot cause a negative loop, contradicting the premise. ■

Computational complexity: The worst-case computational complexity of the new algorithm is conjectured to be exponential with respect to the number of observation time-points and pseudo-polynomial with respect to h. A more precise characterization is under investigation. In the meantime, the following section presents an experimental evaluation of the algorithm across a variety of CSTNs that include worst-case structures to provide some insight into its practical behavior.

IV. Empirical Evaluation

This section presents a preliminary empirical evaluation of our DC-checking algorithm for CSTNs (Algorithm 2). The algorithm and procedures necessary for its evaluation were implemented in Java and executed on a Java Virtual Machine 7 in a PowerBook PC with a 2.2 GHz Intel Core i7 CPU and 4 GB of RAM. The source code is freely available [13].

The main goal was to compute the average computational time for the algorithm across a suite of 60 CSTNs, each having the same number of nodes, but differing with respect to various relevant characteristics. Since randomly generating CSTN instances is the subject of ongoing work, we manually generated 60 networks: 30 *consistent* and 30 *inconsistent.* The CSTNs all had the same size (25 nodes), and each included a negative q-star, but they had different numbers of observation nodes (3-6) and negative q-loops (4-6). Note that these structures represent worst-case instances for our algorithm, especially negative q-loops with one or more non-negative edges.

For the 30 *consistent* instances, the average computational time was 931 milliseconds (ms) with a standard deviation of 1283 ms. For the 30 *inconsistent* instances, the average time was 2140 ± 2166 ms. These results confirm that the new DC-checking algorithm for CSTNs can be practical even for networks exhibiting complex, worst-case structures.

We also considered a set of 60 CSTNs having the same characteristics as in the above set but whose negative q-loops consisted solely of negative edges. Because our algorithm detects and processes these kinds of negative q-loops much more quickly than those having one or more non-negative edges, we expected it to run faster on these instances. Indeed, for the 30 *consistent* instances, the average computational time was 774 ± 838 ms; and for the 30 *inconsistent* instances, the average time was 1494 ± 1658 ms.

Next, to illustrate the potential scalability of our algorithm, we ran it on some consistent CSTNs, each having 100 nodes, 4 negative q-stars, 18 negative q-loops, and 3 observation nodes. For these instances, the average computational time was 1467 ± 175 ms. Figure 6 shows a partial screen-shot of the simulator during the analysis of one such network.

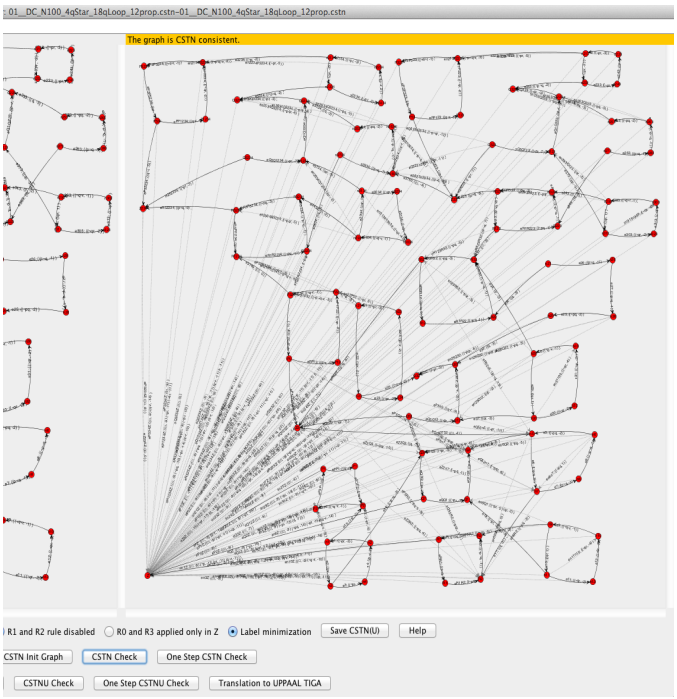


Fig. 6: A screen-shot of a portion of the simulator during the analysis of a CSTN having 100 nodes

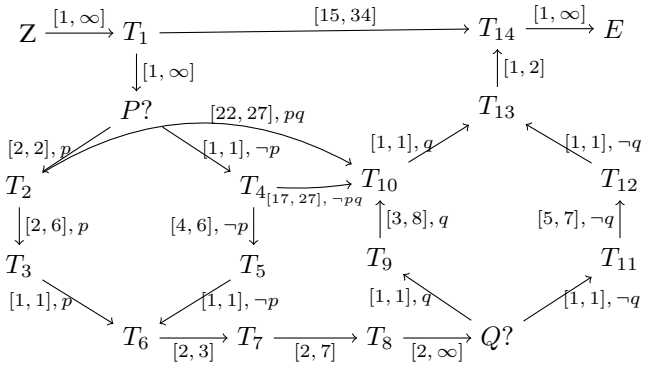


Fig. 7: A CSTN that challenges the TGA-based algorithm

For further perspective on these results, note that the only other existing implementation of a DC-checking algorithm for CSTNs is the TGA-based algorithm [2] which requires about 25 *minutes* $(1,500,000$ ms) to solve the network in Figure 7, whereas our algorithm requires only 386 ± 154 ms on a Linux/Ubuntu machine with an AMD Opteron 4334 processor running at 3 GHz, kernel 3.13.0-37, 62 GB RAM.[8]

V. Conclusions and Future Work

The most significant contribution of this paper is the new propagation-based DC-checking algorithm for CSTNs that is both sound *and complete*. Unlike existing algorithms, an initial evaluation of the performance of this algorithm suggests that it may be practical for a variety of applications. The new algorithm properly addresses the challenges raised by *negative q-loops* and *negative q-stars,* which have never been identified before. Future work will analyze the worst-case complexity; conduct a more intensive empirical evaluation; and aim to extend the same algorithmic techniques to develop a corresponding sound-and-complete algorithm for CSTNUs.

[8]Our software is written in Java 7 and, thus, can be executed on any suitably equipped machine. The TGA-based algorithm exploits a UPPAAL-TIGA procedure (version 1.8) that runs only under Linux or Windows.

References

[1] I. Tsamardinos, T. Vidal, and M. E. Pollack, "CTP: A new constraint-based formalism for conditional, temporal planning," *Constraints*, vol. 8, pp. 365–388, 2003. [Online]. Available: http://dx.doi.org/10.1023/A:1025894003623

[2] A. Cimatti, L. Hunsberger, A. Micheli, R. Posenato, and M. Roveri, "Sound and complete algorithms for checking the dynamic controllability of temporal networks with uncertainty, disjunction and observation," in *21st International Symposium on Temporal Representation and Reasoning (TIME-2014), Verona, Italy*, A. Cesta, C. Combi, and F. Laroussinie, Eds. IEEE Computer Society, September 2014, pp. 27–36. [Online]. Available: http://dx.doi.org/10.1109/TIME.2014.21

[3] P. R. Conrad and B. C. Williams, "Drake: An efficient executive for temporal plans with choice," *Journal of Artificial Intelligence Research (JAIR)*, vol. 42, pp. 607–659, 2011. [Online]. Available: http://dx.doi.org/10.1613/jair.3478

[4] 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 (PlanEx) at ICAPS-2012*, Atibaia, Jun. 2012, pp. 1–8. [Online]. Available: http://arxiv.org/abs/1212.2005

[5] 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. Fred, Eds., vol. 2. SCITEPRESS, Feb. 2013, pp. 144–156.

[6] ——, "An algorithm for checking the dynamic controllability of a conditional simple temporal network with uncertainty - revisited," in *Agents and Artificial Intelligence*, ser. Communications in Computer and Information Science, J. Filipe and A. Fred, Eds. Verlag Springer, 2014, vol. 449, pp. 314–331. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-44440-5_19

[7] P. H. 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-01)*, B. Nebel, Ed. Kaufmann, 2001, pp. 494–502.

[8] R. Dechter, I. Meiri, and J. Pearl, "Temporal constraint networks," *Artificial Intelligence*, vol. 49, no. 1-3, pp. 61–95, 1991. [Online]. Available: http://dx.doi.org/10.1016/0004-3702(91)90006-6

[9] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall, 2010.

[10] P. Morris, "A structural characterization of temporal dynamic controllability," in *Principles and Practice of Constraint Programming (CP 2006)*, ser. LNCS, F. Benhamou, Ed., vol. 4204. Springer, 2006, pp. 375–389. [Online]. Available: http://dx.doi.org/10.1007/11889205_28

[11] P. H. Morris and N. Muscettola, "Temporal dynamic controllability revisited," in *The 20th Nat. Conf. on Art. Intell. (AAAI-05)*, M. M. Veloso and S. Kambhampati, Eds. AAAI Press, 2005, pp. 1193–1198.

[12] E. Schwalb and R. Dechter, "Processing disjunctions in temporal constraint networks," *Artificial Intelligence*, vol. 93, no. 1-2, pp. 29–61, 1997.

[13] R. Posenato, "A CSTN(U) Consistency Check Algorithm Implementation in Java," http://profs.scienze.univr.it/~posenato/software/cstnu, May 2015.