Consistency Checking of STNs with Decisions: Managing Temporal and Access-Control Constraints in a Seamless Way

Matteo Zavatteri[a,*], Carlo Combi[a], Romeo Rizzi[a], Luca Viganò[b]

[a] *Department of Computer Science, University of Verona, Verona, Italy*
[b] *Department of Informatics, King's College London, London, UK*

Abstract

A Simple Temporal Network (STN) consists of time points modeling temporal events and constraints modeling the minimal and maximal temporal distance between them. A Simple Temporal Network with Decisions (STND) extends an STN by adding *decision time points* to model temporal plans with decisions. A decision time point is a special kind of time point that once executed allows for deciding a truth value for an associated Boolean proposition. Furthermore, STNDs label time points and constraints by conjunctions of literals saying for which *scenarios* (i.e., complete truth value assignments to the propositions) they are relevant. In this paper, we deal with the use of STNDs for modeling and synthesizing execution strategies on real world planning problems, by considering some motivating scenarios from the area of healthcare processes. More precisely, we focus on the issue of checking the consistency of STNDs and propose an incremental hybrid SAT-based consistency checking algorithm for STNDs that (i) is faster than the one previously proposed and (ii) allows for the synthesis of all consistent scenarios and related early execution schedules (offline temporal planning). We employ STNDs to model and reason about both temporal and access-control constraints. We carry out an experimental evaluation with Kappa, a tool that we developed for STNDs. Finally, as a last contribution from the theoretical side, since consistency of STNDs is equivalent to consistency of

*Corresponding author
Email addresses: matteo.zavatteri@univr.it (Matteo Zavatteri), carlo.combi@univr.it (Carlo Combi), romeo.rizzi@univr.it (Romeo Rizzi), luca.vigano@kcl.ac.uk (Luca Viganò)

disjunctive temporal networks (DTNs), as for both formalisms consistency is NP-complete, we show how any STND can be easily translated into a DTN and vice versa.

Keywords: Simple temporal network with decisions, HSCC algorithms, incremental SAT-solving, disjunctive temporal network, temporal constraints, access control

---

1. Introduction

Temporal constraint networks are a set of formalisms that allow one to model temporal plans and check the coherence of temporal constraints that impose lower and upper bounds on the temporal distance of the modeled events. Different kinds of temporal constraint networks have been proposed, according to the nature of temporal constraints and temporal features. *Simple Temporal Networks* (*STNs*, [1]) are the common ground, i.e., the basic formalism, used in many extensions [2, 3, 4]. STNs are able to model an unconditional temporal plan in which all components (events and their time distances) are under control. Temporal events are modeled as *time points* and their occurrence is modeled by executing the corresponding time points (i.e., by assigning them real values).

A temporal plan is *consistent* if we can schedule all the events, each at a specific time instant, such that all constraints are satisfied. If this is possible, a schedule can be synthesized for both offline planning (the plan is available before starting) or online planning (the plan is generated while executing the given time points). The difference between these two planning approaches is that in the former the consistency checking algorithm returns a solution, whereas in the latter the algorithm returns a *minimal network* to generate any possible solution.

Recently, the interest of the business process research community for temporal constraint networks increased. Such formalisms have been considered the most suitable solution for mapping and then formally verifying temporal properties of *business process models*, where temporal constraints are specified with respect to the execution of tasks composing such process models [5, 6, 7, 8, 9]. In this case, plans are related to the execution of process paths, possibly composed by different tasks, according to different choices or conditions holding during the process execution. Such plans are also usually completed by the task assignments to suitable agents, accord-

ing to some given access-control requirements. Access-control requirements, often expressed by *Role-Based Access Control (RBAC)* models [10], are related to the issue of assigning tasks to different authorized agents, satisfying some given constraints. Such constraints are often intertwined with temporal ones [11]. Thus, consistency could be related both to temporal and access-control features.

Furthermore, temporal planning is also relevant in the field of project planning [12, 13], which is quite close to that of business processes. Here, temporal requirements are always specified and often intertwined with resource-allocation (or access control) constraints and with different decisions about the actions to execute [14, 15].

According to this wide perspective, STNs fail to model temporal plans where the occurrence of some events or the satisfaction of some constraints must happen only if some decision has been made. The decisions we are interested in have Boolean domain. Thus, a temporal plan subject to decisions would ask us (not) to execute some time points or to satisfy some constraints depending on what decisions we have made.

A few proposals to handle decisions within the temporal network formalisms built on top of STNs have been put forth. For instance, *Drake* [16] provides an executive for temporal plans with choices based on *Labeled STNs* that do not specify decision points (in a node-sense), whereas *Temporal Plan Networks* (*TPNs*, [17]) extend STNs with decision nodes and model decisions as outgoing edges from such nodes. *Simple Temporal Networks with Decisions* (*STNDs*, [18, 19]) extend STNs by adding decision points that can influence both the execution of time points and the satisfaction of constraints. Several (possibly different) STNs may arise when projecting an STND onto a scenario that models the complete interpretation of the decisions. For any scenario, we are only interested in executing the time points and satisfy the constraints *entailed* by it.

So far, only one hybrid SAT-based consistency checking algorithm (*HSCC*) has been devised to check the consistency of an STND [18]. If the algorithm finds a consistent scenario, i.e., a scenario for which the STN projection is consistent, then a solution (*scenario* plus *schedule*) exists. In this case, any schedule in the solution set involves the STN-projection corresponding to the related scenario.

This algorithm allows for an offline planning where all decisions are made before starting and the corresponding schedule is already known. However, this algorithm has never been implemented nor evaluated, and it does not

allow for the synthesis of all consistent scenarios.

In this paper, we extend and complete the first, preliminary, proposal dealing with consistency checking of STNDs that we gave in [20]. Here, we provide a deeper discussion and motivation of our proposal, by describing requirements from the area of healthcare business processes having temporal constraints. More specifically, we first focus on modeling and checking through STNDs the consistency of business processes with respect to some given temporal constraints. We then discuss how to use STNDs to model, represent, and check requirements related to the access control for temporal business processes. We also add further details about the implementation, verification and synthesis of execution strategies of the provided motivating scenario.

The main contributions of our proposal can be summarized as follows:

1. As a proof of concept, we introduce and discuss requirements from a real world scenario related to healthcare processes. It motivates the study and the improvements proposed for STNDs both for modeling, validating and synthesizing execution plans, satisfying both temporal and access-control constraints.
2. We introduce and discuss `STND-HSCC2`, an HSCC algorithm for STNDs that (i) is faster than the existing algorithm as it rules out inconsistent scenarios as early as possible and (ii) allows for the synthesis of all consistent scenarios and related early execution schedules (offline temporal planning). The algorithm is hybrid because a SAT-solver and a shortest path algorithm mutually influence each other (the output of the former becomes the input of the latter and vice versa continuously).
3. We propose and discuss an original approach to represent access-control requirements through STNDs, and to manage such requirements, possibly also temporal ones, together with other pure temporal constraints, in a seamless way.
4. We discuss Kappa, a tool that we developed for the experimental evaluation of STNDs. As a minor contribution, we adapted the previous algorithm to support the synthesis of all consistent scenarios.
5. As a final contribution from the theoretical point of view, we consider *disjunctive temporal networks (DTNs)* and show how to directly translate any STND into a corresponding DTN and vice versa. DTNs have been introduced to represent disjunctive temporal constraints [21].

Thus, similarly to STNs, they allow for the representation of different scenarios, according to the constraint considered in any given disjunction. As these two representations and reasoning formalisms are equivalent at complexity level, it is important to understand, from the modeling point of view, how to move from one to the other and discuss whether such kind of mappings are easily understandable when representing real world networks.

The paper is organized as following. Section 2 introduces the concrete healthcare case study that we consider throughout the paper. Section 3 provides background on STNs and STNDs, and the current consistency checking algorithm for STNDs. Section 4 discusses `STND-HSCC2`, a faster HSCC algorithm for STNDs. Section 5 shows how STNDs can be used to represent and verify access-control constraints. Section 6 discusses Kappa and the related experimental evaluation. Section 7 provides polynomial-time reductions to translate any STND into a DTN and vice versa. Section 8 discusses related work. Section 9 sums up and discusses future work.

2. A Case Study from the Healthcare Domain

To motivate and illustrate our proposal, we focus on the domain of stress test in the clinical domain of cardiology and applied physiology [22]. Figure 1 depicts a simple process in which a patient has to undergo a cardiological visit (CardioVisit); then, two possible mutually exclusive electrocardiographic (ECG) stress tests have to be executed according to different techniques, i.e., treadmill (Treadmill-StressECG) or cycle ergometer (CycleErgom-StressECG); after the ECG stress test, the results are analyzed and a blood test is performed according to two possibly different types of lab device; finally, the overall final report is completed (FinalReport). The overall process has to be completed within 90 to 156 minutes, while the time span admitted between the beginning of the cardiological visit and the end of the analysis of stress test results is between 35 and 45 minutes.

To represent graphically the process, we adopted the well-known *BPMN language* [23]. Figure 1 shows the BPMN process model for cardiological patients. The undergone activities correspond to *tasks*, displayed as rounded boxes containing the name of the activities. The alternative possible paths of execution are represented through diamonds corresponding to either XOR split or join gateways. The choice of the path to follow during a specific

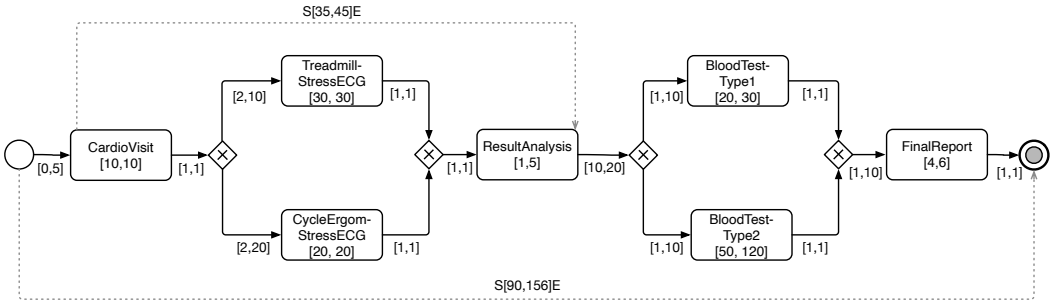


Figure 1: The BPMN process model for cardiological patients.

execution of the stress test process can be done by the software engine (i.e., business process engine) that manages the coordinated activations of tasks. The start and end of the process are represented through circles with a single outgoing and a single ingoing edge, respectively. Edges between tasks/gateways represent a precedence relation with respect to their execution. For example, the edge between task CardioVisit and the following XOR split means that the cardiological visit has to be finished before continuing with the XOR split. The BPMN representation has been enriched with some temporal annotations. Tasks and edges have been annotated by ranges $[l, u]$ of time units needed either to execute the task or to start another task/gateway after the end of the previous task/gateway, respectively. For example, ResultAnalysis can span from 1 to 5 time units (i.e., minutes). Finally, to represent graphically the temporal constraints between tasks, we enriched the BPMN model with labeled dotted lines linking the considered BPMN constructs. For example, the allowed time span between the start of task CardioVisit and the end of ResultAnalysis is expressed through the notation $S[35, 45]E$.

According to the depicted example, the following questions arise.

- Are we able to understand whether the specified temporal constraints are meaningful? In other words, is it possible to establish whether there is some execution of the specified process that satisfies all the given constraints?
- Is it possible to establish which execution paths are allowed with respect to the possible alternative options the software engine has at disposal?
- Is there some meaningful execution of the process using the cycle ergometer ECG stress test?

These three questions are obviously intertwined but have some differences also from the application point of view. Indeed the first one refers to the existence of a solution, the second one to the discovery of all the allowed execution plans (i.e., satisfying all the given constraints), and the third one refers to the feasibility of a set of specific plans, all containing a given task.

To answer these questions, the BPMN model is usually mapped into a corresponding temporal constraint network. Different kinds of temporal constraint networks have been proposed, according to the nature of temporal constraints and temporal features of tasks and gateways of process models [6, 7, 9]. In this paper, we focus on a mapping to STNDs. Such a new mapping, which is specified similarly to the mappings proposed in literature with respect to other kinds of temporal constraint networks, allows for a suitable management of different execution paths that are under the control of the software engine, whereas in the previous proposals the focus was on alternative paths (and task durations) that were uncontrollable, i.e., determined according to some conditions depending on external entities.

Informally, the mapping of a process model into an STND is done by considering each single construct. More precisely, each task is mapped to two corresponding time points, one for the starting time and the other one for the ending time of the task, respectively. The allowed timespan of the task is expressed through two temporal constraints. XOR split gateways are expressed as single decision time points. XOR join gateways, as well as start and end of the process, are represented as single time points. Temporal constraints between the components in the process are mapped to temporal constraints between the corresponding time points. Both time points and constraint edges are suitably labeled according to the scenarios they belong to. The STND in Figure 3a encodes the process in Figure 1.

After the specification of temporal constraints, let us slightly extend our case study. We would like to consider some further, non temporal constraints, related to the agents allowed to execute the different tasks of the process.

The requirement is that tasks CardioVisit, ResultAnalysis, and FinalReport have to be executed by a cardiologist, while BloodTestType1 and BloodTestType2 have to be executed by a physician specialized in Laboratory Medicine. Moreover, moving to more strict requirements, the cardiologist executing CardioVisit must also execute ResultAnalysis and FinalReport, for a given process instance. On the other side, the same physician cannot execute both CardioVisit and BloodTestType1/BloodTestType2 in a single process instance.

Kate and Mike are two physicians both specialized in Cardiology and in Laboratory Medicine. Thus, they could potentially execute CardioVisit, ResultAnalysis, FinalReport, and BloodTestType1/BloodTestType2.

According to these further details related to access control, other issues arise.

- Is it possible to specify task assignments to Kate and Mike for the execution of a single process instance, satisfying all the given constraints?
- Is it possible that the same physician has assignments related to both medical specialties Cardiology and Laboratory Medicine in a given process execution?

Such issues are related to access-control requirements, faced in many contributions according to different Access Control models, often considering roles in specifying access constraints [10, 24]. The specified requirements/-constraints are not temporal and, thus, they can be checked in an orthogonal way with respect to the previously introduced temporal ones. However, it is straightforward to consider possible temporal access-related constraints, where access-control and temporal requirements are intertwined. As an example, let us assume that we want to consider a further constraint: if Mike executes BloodTest-Type2, the overall process must end within 100 minutes.

Thus, a new question arises.

- Is it possible that Mike executes BloodTest-Type2 and the given process ends without violating any given temporal constraint?

Figure 2 depicts the BPMN process related to the management of cardiological patients that we previously discussed for the temporal aspects, enriched with BPMN artifacts named *annotations* [23] to specify access-control constraints as well as temporal access-control constraints. As we did for temporal constraints, also access-control constraints will be mapped from BPMN artifacts to suitable STND fragments, as we will discuss in Section 5.

3. Simple Temporal Network with Decisions

In this section, we will briefly introduce STNs, then will focus on STNDs, discuss the basic related concepts and, at the end, discuss suitable algorithms for checking the consistency of STNDs.

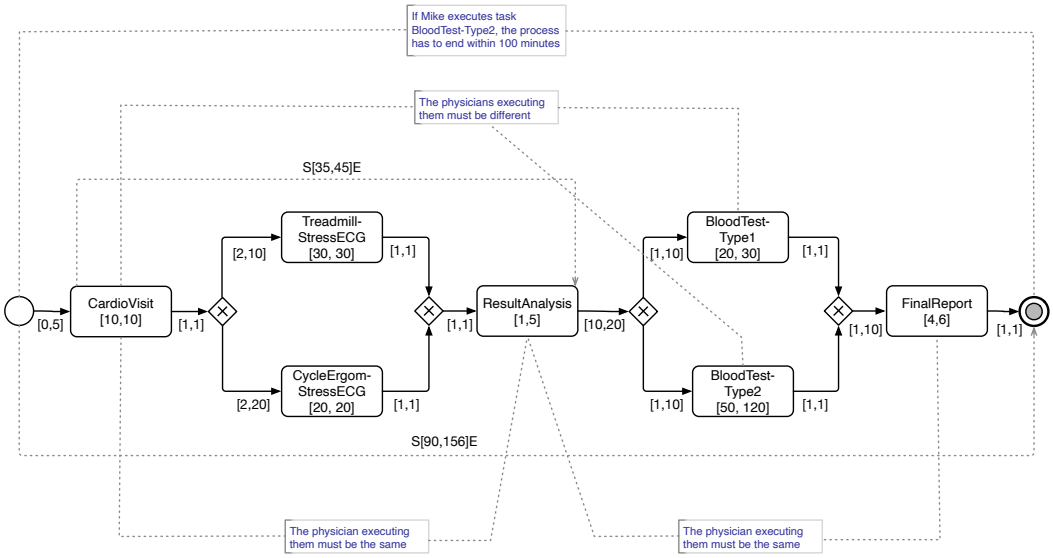


Figure 2: The BPMN process model for cardiological patients extended with some access constraints.

A *Simple Temporal Network* (*STN*, [1]) is a pair $\langle \mathcal{T}, \mathcal{C} \rangle$, where $\mathcal{T} = \{X, \dots\}$ is a set of time points (continuous variables) and $\mathcal{C} = \{(Y - X \leq k), \dots\}$ is a set of constraints, for $X, Y \in \mathcal{T}$, $k \in \mathbb{R} \cup \pm\{\infty\}$. An STN is *consistent* if there exists a schedule $t \colon \mathcal{T} \to \mathbb{R}$ of real values to the time points such that all constraints are satisfied. Given a consistent STN, a *schedule* must enforce that if X is executed before Y, then $t(X) \leq t(Y)$. An *early execution* of an STN consists of finding a schedule executing the time points as soon as possible (e.g., by using the Floyd-Warshall algorithm [1]).

Given a set $\mathcal{P} = \{d, \dots\}$ of Boolean propositions, a *label* $\ell = \lambda_1 \dots \lambda_n$ is any finite conjunction of literals λ_i, where a literal is either d (positive literal) or $\neg d$ (negative literal), and we might sometimes omit the $\wedge$ connective to ease reading when the label is clear from the context. The *empty label* is denoted by $\boxdot$. The *label universe of* $\mathcal{P}$, denoted by $\mathcal{P}^*$, is the set of all possible (consistent) labels drawn from $\mathcal{P}$; e.g., if $\mathcal{P} = \{d, e\}$, then $\mathcal{P}^* = \{\boxdot, d, e, \neg d, \neg e, de, d\neg e, \neg de, \neg d\neg e\}$. Two labels ℓ_1, ℓ_2 are *consistent* if their conjunction $\ell_1\ell_2$ is satisfiable. A label ℓ_1 *entails* a label ℓ_2 (written $\ell_1 \Rightarrow \ell_2$) if all literals in ℓ_2 appear in ℓ_1 too (i.e., if ℓ_1 is more *specific* than ℓ_2). For instance, if $\ell_1 = d\neg e$ and $\ell_2 = d$, then ℓ_1 and ℓ_2 are consistent since $d\neg ed$ is satisfiable, and ℓ_1 entails ℓ_2 since $d\neg e \Rightarrow d$.

A *scenario* is a mapping $s \colon \mathcal{P} \to \{\top, \bot\}$ assigning a truth value to each

$d \in \mathcal{P}$. A scenario *satisfies* a label ℓ (in symbols $s \models \ell$) if ℓ evaluates to true under the interpretation given by s (e.g., if $s(d) = \top$ and $s(e) = \bot$, then $s \models d\neg e$).

Definition 1. A *Simple Temporal Network with Decisions* (*STND*, [18, 19, 25]) is a tuple $\mathcal{S} = \langle \mathcal{T}, \mathcal{DT}, \mathcal{P}, O, L, \mathcal{C} \rangle$, where:

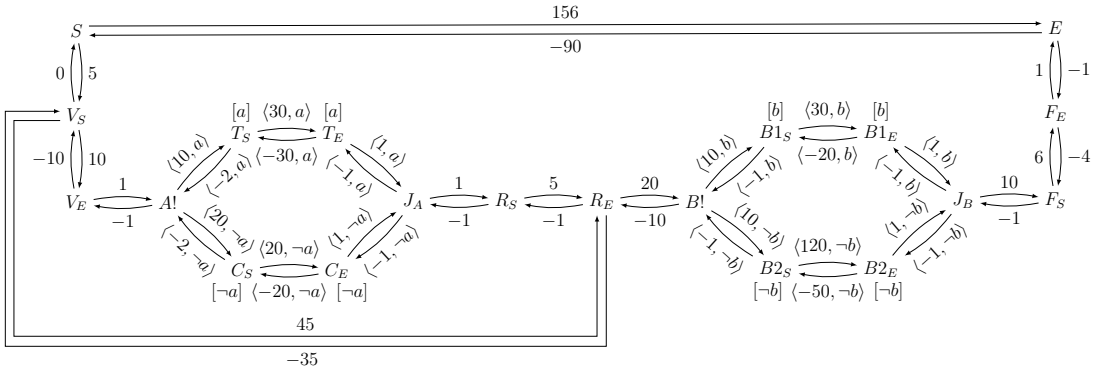
- $\mathcal{T} = \{X, \ldots, Z\}$ is a finite set of time points.
- $\mathcal{DT} \subseteq \mathcal{T} = \{D!, \ldots, H!\}$ is a finite set of decision time points.
- $\mathcal{P} = \{d, \ldots, g\}$ is a finite set of Boolean propositions.
- $O \colon \mathcal{P} \to \mathcal{DT}$ is a bijection assigning a unique proposition to each decision time point $D!$ that controls the truth value assignment to d ($O^{-1} \colon \mathcal{DT} \to \mathcal{P}$ models the inverse).
- $L \colon \mathcal{T} \to \mathcal{P}^*$ is a function assigning labels to time points.
- $\mathcal{C}$ is a finite set of *labeled* constraints $(Y - X \leq k, \ell)$ where $X, Y \in \mathcal{T}$, $k \in \mathbb{R} \cup \pm\{\infty\}$ and $\ell \in \mathcal{P}^*$.

The STN-projection of an STND $\mathcal{S}$ with respect to a scenario s (written $\pi_s(\mathcal{S})$) is an STN $\langle \mathcal{T}_s, \mathcal{C}_s \rangle$ built as follows:

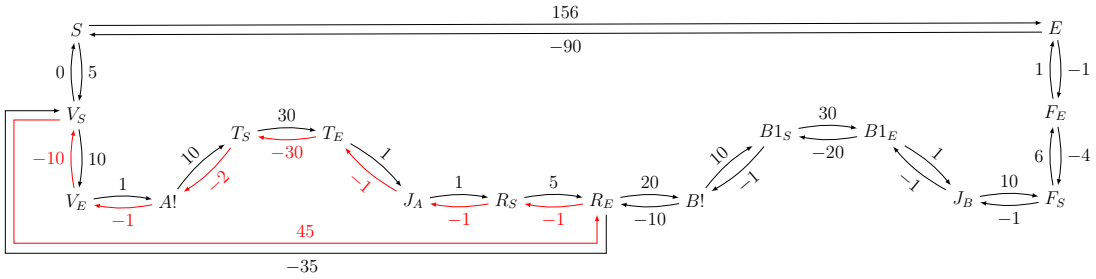
- $\mathcal{T}_s = \{X \mid X \in \mathcal{T} \wedge s \models L(X)\}$
- $\mathcal{C}_s = \{(Y - X \leq k) \mid (Y - X \leq k, \ell) \in \mathcal{C}, s \models \ell\}$

$\mathcal{S}$ is consistent if there exists a scenario s such that $\pi_s(\mathcal{S})$ is consistent. A solution is a pair $\langle s, t \rangle$, where s is a scenario, t is a schedule with domain $\mathcal{T}_s$, and $t(Y) - t(X) \leq k$ holds for each $(Y - X \leq k) \in \mathcal{C}_s$. Checking consistency of STNDs is NP-complete [18].

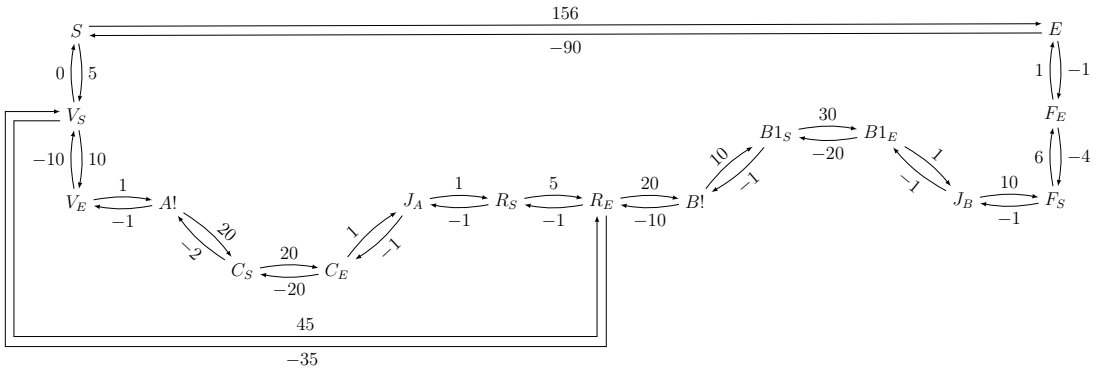
We represent an STND graphically by extending the distance graph of an STN into a labeled distance (multi)graph. The set of nodes still coincides with the set of time points, whereas each edge $X \to Y$ labeled by $\langle k, \ell \rangle$ represents $(Y - X \leq k, \ell) \in \mathcal{C}$ (when $\ell = \boxdot$, we just use k as a label). Negative values on edges model delays, positive ones model deadlines (when their labels are true). Time points' labels, when different from $\boxdot$, are shown near the nodes between $[\ldots]$ brackets. Many $\langle k, \ell \rangle$ can be specified for the



(a) STND modeling the business process of Figure 1.



(b) Projection of Figure 3a onto $s(a) = \top$ and $s(b) = \top$. The resulting (inconsistent) STN models the process path of Figure 1 going through Treadmill-StressECG and BloodTest-Type1.



(c) Projection of Figure 3a onto $s(a) = \bot$ and $s(b) = \top$. The resulting (consistent) STN models the process path of Figure 1 going through CycleErgom-StressECG and BloodTest-Type1.

Figure 3: An example of an STND and two of its (four) possible STN-projections.

same $X \to Y$ provided their ℓ are different (if two labels are equal, we keep the smallest k). Figure 3a shows an example of an STND, whereas

Figures 3b–3c show two of its (four) possible STN-projections.

A label ℓ labeling a time point or a constraint is *honest* if for each literal d or $\neg d$ in ℓ we have that $\ell \Rightarrow L(D!)$, where $D! = O(d)$ is the decision time point associated to d; ℓ is dishonest otherwise. A label on a constraint is *coherent* if it is at least as expressive as the labels of the time points appearing in the constraint (i.e., ℓ contains all literals in the labels of the two connected time points).

Definition 2 (Well-definedness). An STND is well-defined [19, 25, 26] if

- $L(X) \Rightarrow L(O(d))$ and $(O(d) - X \leq 0) \in \mathcal{C}$ for each $X \in \mathcal{T}$ and $\{d, \neg d\} \in L(X)$, and
- $\ell \Rightarrow L(Y) \wedge L(X)$ for each $(Y - X \leq k, \ell) \in \mathcal{C}$, and $\ell \Rightarrow L(O(d))$ for each literal $\{d, \neg d\} \in \ell$.

The STND in Figure 3a is well-defined.

To check the consistency of an STND, we can use the *hybrid SAT-based consistency checking (HSCC)* algorithm proposed in [18]. `STND-HSCC1`, specified in Algorithm 1, (`STND-CC` in [18]) maintains a formula φ specifying CNF clauses over propositions in $\mathcal{P}$. Initially, φ allows for all truth value assignments. In each round of the algorithm we ask the SAT solver for a truth value assignment making φ true. Such an assignment (if any) corresponds to a scenario s over which we can project the STND and check if the resulting STN is consistent ("SAT-solver influences directed weighted graph algorithm"). If so, we return this scenario and a valid schedule for the projected STN (i.e., a solution). Otherwise, we apply De Morgan's rules to the negation of the *relevant part* of the scenario containing the negative cycle (`CUT-SCENARIO` in Algorithm 2) and add the resulting clause to φ and go ahead with the next round ("directed weighted graph algorithm influences the SAT-solver"). This makes the approach hybrid. If φ has become unsatisfiable, it means that all STN-projections are inconsistent and therefore the STND is inconsistent. Similar approaches are described in [27, 28].

An example of round for the network in Figure 3a is as follows. Suppose that `SAT-SOLVE`$(\varphi) = ab$ (i.e., $s(a) = \top$ and $s(b) = \top$). Since the STN $\pi_{ab}(\mathcal{S})$ is inconsistent (Figure 3b admits a negative cycle), we add to φ the clause $\neg(a \wedge b)$, which simplifies to $(\neg a \vee \neg b)$, to ask the SAT solver for a different truth value assignment excluding this projection (if any). Figure 3b is consistent if and only if $s(a) = \bot$ and $s(b) \in \{\top, \bot\}$. Consider, for

Algorithm 1: STND-HSCC1($\mathcal{S}$)

Input: An STND $\mathcal{S} = \langle \mathcal{T}, \mathcal{DT}, \mathcal{P}, O, L, \mathcal{C} \rangle$.

Output: A solution $\langle s, t \rangle$ for $\mathcal{S}$ if $\mathcal{S}$ is consistent; "*inconsistent*" if $\mathcal{S}$ is inconsistent. STN-CC is a consistency checking algorithm for STNs.

```
1 φ ← ⋀_{p∈P}(p ∨ ¬p)                          ▷ Make every assignment possible
2 while true do
3     s ← SAT-SOLVE(φ)                          ▷ Try to find a satisfying assignment s
4     if φ is unsatisfiable then
5         return inconsistent
6     ⟨T_s, C_s⟩ ← π_s(S)                       ▷ Project S onto s
7     if STN-CC(⟨T_s, C_s⟩) then
8         return ⟨s, t⟩                         ▷ where S is an early schedule for ⟨T_s, C_s⟩
9     φ ← φ ∧ CUT-SCENARIO(S, ⟨T_s, C_s⟩)       ▷ Exclude this scenario
```

Algorithm 2: CUT-SCENARIO($\mathcal{S}, \langle \mathcal{T}_s, \mathcal{C}_s \rangle$)

Input: An STND $\mathcal{S} = \langle \mathcal{T}, \mathcal{DT}, \mathcal{P}, O, L, \mathcal{C} \rangle$ and one of its STN-projections $\langle \mathcal{T}_s, \mathcal{C}_s \rangle$

Output: A clause ψ expressing the cut of the relevant part of the scenario.

```
1 ψ ← ⊤                                        ▷ ψ will contain the relevant part of s
2 foreach constraint c ∈ C_s do
3     ψ ← ψ ∧ ℓ_c                              ▷ where ℓ_c is the corresponding label of c in S
4 return DeMorgan(¬(ψ))                        ▷ the clause expressing the cut of ψ
```

instance, the scenario $s(a) = \bot$ and $s(b) = \top$ shown in Figure 3c. A possible schedule for Figure 3c is $t(S) = 0$, $t(V_S) = 0$, $t(V_E) = 10$, $t(A) = 11$, $C_S = 13$, $C_E = 33$, $J_A = 34$, $R_S = 35$, $R_E = 36$, $B = 46$, $B1_S = 47$, $B1_E = 72$, $J_B = 73$, $F_S = 83$, $F_E = 89$ and $E = 90$.

4. A Faster HSCC Algorithm for STNDs

STND-HSCC1 is correct [18], but it suffers from the limitation that projections are tested only when the SAT solver returns a complete truth value assignment. Consider Figure 3a and assume that the SAT solver starts on the formula $\varphi = (a \vee \neg a) \wedge (b \vee \neg b)$, which makes every truth value assign-

ment possible. Suppose that in the search tree the SAT solver decides $\bot$ for a proposition d going down to the left and $\top$ going down to the right in the search tree, and assume that the order of visit is right then left. The first truth value assignment returned is $a = \top$ and $b = \top$ (corresponding to the scenario $s(a) = s(b) = \top$). Now `STND-HSCC1` would project the STND of Figure 3a onto s to obtain the STN shown in Figure 3b and eventually detect the negative cycle. However, the negative cycle could have been detected much earlier, say, when a was assigned $\top$. Indeed, all projections of any scenario containing $s(a) = \top$ generate an STN embedding a negative cycle between V_S, V_E, $A!$, T_S, T_E, J_A, R_S and R_E, no matter which Boolean value is assigned to b (Figure 3b, red cycle). Therefore, a clever implementation of this algorithm calls for an early detection of negative cycles. Before proceeding with it, we must refine the concept of scenario and projection so that they support "unknown" propositions (i.e., propositions that have not been assigned a value yet).

Definition 3. A *scenario* is (now) a mapping $s\colon \mathcal{P} \to \{\top, \bot, -\}$ assigning either *true*, *false* or *unknown* to each proposition $d \in \mathcal{P}$. A scenario s *satisfies* a label ℓ if ℓ evaluates to true under the following interpretation given by s:

1. $s \models \lambda$ iff $(\lambda = d \wedge s(d) = \top)$ or $(\lambda = \neg d \wedge s(d) = \bot)$,
2. $s \models \ell$ iff $s \models \lambda_1$ and $\ldots$ and $s \models \lambda_n$ for $\ell = \lambda_1 \ldots \lambda_n$.

Note that s never satisfies a label containing a literal for which the corresponding proposition is unknown in s.

The definition of STN projection remains the same as that given in Definition 1 but extended to the new definition of scenario.

`STND-HSCC2` (Algorithm 3) is a brand new algorithm to check the consistency of STNDs. It allows for the synthesis of either a single or all scenarios admitting a consistent schedule for the corresponding STN-projection. `STND-HSCC2` still initializes a CNF formula φ making all truth value assignments possible. Then, it starts the SAT-solver and hooks a listener to the corresponding run. Such a listener is able to operate on φ by adding CNF clauses on the fly if needed and is triggered by two main events: *assume* and *solution found*.

An *assume* ($d = \top$ or $d = \bot$) event (Algorithm 3, lines 8-12) triggers an action of the listener to "look ahead" if the STN-projection obtained by projecting the STND onto the scenario built from the current truth value

Algorithm 3: `STND-HSCC2`$(\mathcal{S}, all)$

Input: An STND $\mathcal{S} = \langle \mathcal{T}, \mathcal{DT}, \mathcal{P}, O, L, \mathcal{C} \rangle$ and a Boolean value *all* meaning *all consistent scenarios* iff $all = \top$.

Output: A single or all scenarios s along with schedule(s) t for the projection $\pi_s(\mathcal{S})$ if $\mathcal{S}$ is consistent, "*inconsistent*" otherwise.

1 $\varphi \leftarrow \bigwedge_{p \in \mathcal{P}} (p \vee \neg p)$ ▷ Make every assignment possible
2 $Sols = \emptyset$ ▷ The set of (all) consistent scenarios (global variable)
3 Hook a *listener* to the run of the SAT solver and make it detect negative cycles as early as possible (**on** blocks (below) define the event-driven behavior).
4 `SAT-SOLVE`(φ)
5 **if** $Sols = \emptyset$ **then**
6 **return** *inconsistent*
7 **return** *Sols*
8 **on** *assume* $d = \top$ *or assume* $d = \bot$*:* ▷ Partial model
9 Build a scenario s from the current truth value assignment extended with $s(d) = \top$ or $s(d) = \bot$ (depending on the case)
10 $\langle \mathcal{T}_s, \mathcal{C}_s \rangle \leftarrow \pi_s(\mathcal{S})$ ▷ Get the STN projection
11 **if** *BellmanFord*$(\langle \mathcal{T}_s, \mathcal{C}_s \rangle)$ *detects a negative cycle* **then**
12 $\varphi \leftarrow \varphi \wedge$ `CUT-SCENARIO`$(\mathcal{S}, \langle \mathcal{T}_s, \mathcal{C}_s \rangle)$ ▷ Add clause
13 **on** *solution found:* ▷ Complete model
14 Build a scenario s from the current truth value assignment
15 $\langle \mathcal{T}_s, \mathcal{C}_s \rangle \leftarrow \pi_s(\mathcal{S})$ ▷ Get the STN projection
16 **if** *BellmanFord*$(\langle \mathcal{T}_s, \mathcal{C}_s \rangle)$ *does not detect any negative cycle* **then**
17 $Sols \leftarrow Sols \cup \{\langle s, t \rangle\}$ ▷ t is an early schedule for $\langle \mathcal{T}_s, \mathcal{C}_s \rangle$
18 **if** *BellmanFord*$(\langle \mathcal{T}_s, \mathcal{C}_s \rangle)$ *detects negative cycle or all is true* **then**
19 $\varphi \leftarrow \varphi \wedge$ `CUT-SCENARIO`$(\mathcal{S}, \langle \mathcal{T}_s, \mathcal{C}_s \rangle)$ ▷ Add clause

assignment and *extended* with this assumption contains a negative cycle. If so, we extend φ by adding (on the fly) a CNF clause modeling the negation of the part of s containing a negative cycle in order to avoid getting the same scenario again. If the projection is consistent, `STND-HSCC2` does nothing and lets the run go.

A *solution found* event (Algorithm 3, lines 13-19) extends the behavior of the listener described for *assume* as follows. When triggered, the listener builds a scenario from the current truth value assignment (which does not

need to be extended with anything else this time). Then, it checks if the corresponding STN-projection contains a negative cycle. If it does not, then it computes an (early) schedule t for the STN projected onto s and adds the pair $\langle s, t \rangle$ to the set of solutions. If it detects a negative cycle (or all consistent scenarios are sought), then it acts as for *assume* events.

Eventually, when the run of the SAT solver ends, either $Sols = \emptyset$ (and thus the starting STND is inconsistent), or *Sols* contains at least 1 solution (scenario-schedule).

Like `STND-HSCC1`, `STND-HSCC2` is sound and complete because it is based on a SAT-solver that allows us to iterate on all the models. Whenever we add a clause, we exclude a relevant part of a scenario that we do not want to get anymore. The sooner, the better.

Besides the SAT solver, all the other internal sub-procedures (mostly, algorithms for directed weighted graphs) are well known to be sound and complete, and run in polynomial time.

5. Modeling Access Control

Above, we showed how a business process, augmented with temporal constraints, can be encoded into an STND to check its consistency and synthesize one or all possible schedules. This section proves that we can do more than just modeling time and process paths. This section proves that we can exploit STNDs' decisions to model, check consistency and synthesize schedules of a process augmented with resources, constraints over their commitment and also mixed constraints involving both resources and time. In this section, we summarize the well-consolidated model of RBAC, consider an access control layer for our process and then describe a polynomial-time encoding into STNDs.

Definition 4 (RBAC). A *role-based access control model* is a tuple (U, R, T, UA, TA), where U is a finite set of users, R is a finite set of roles, T is a finite set of tasks, $UA \subseteq U \times R$ is a many-to-many user assignment relation, and $TA \subseteq T \times R$ is a many-to-many task assignment relation. Thus, $(u, r) \in UA$ means that user u belongs to role r, whereas (t, r) means that task t can be executed by any user (i.e., agent) belonging to role r.

The RBAC model of Figure 2 is $U = \{\mathsf{kate}, \mathsf{mike}\}$, $R = \{\mathsf{cardiologist}, \mathsf{pathologist}\}$, $T = \{S, \mathsf{CardioVisit}, \mathsf{ResultAnalysis}, \mathsf{Treadmill\text{-}StressECG},$

CycleErgom-StressECG, BloodTest-Type1, BloodTest-Type2, FinalReport, $E\}$ (S and E model the start and end of the process), $UA = U \times R$, and $TA = \{$(CardioVisit, cardiologist), (ResultAnalysis, cardiologist), (FinalReport, cardiologist), (BloodTest-Type1, pathologist), (BloodTest-Type2, pathologist)$\}$. To simplify, we neglected XOR splits and joins; we also neglected the role patient, because it would not add anything.

Given an RBAC model (U, R, T, UA, TA), we write $users(r) = \{u \mid (u, r) \in UA\}$ for the set of users belonging to a role r and $roles(t) = \{r \mid (t, r) \in TA\}$ for the set of roles authorized for a task t. We also take the liberty of writing $users(t) = \{u \mid (u, r) \in UA, r \in roles(t)\}$ for the set of users authorized for a generic task t. Therefore, $users(\mathsf{CardioVisit}) = users(\mathsf{ResultAnalysis}) = users(\mathsf{BloodTest\text{-}Type1}) = users(\mathsf{BloodTest\text{-}Type2}) = users(\mathsf{FinalReport}) = \{\mathsf{kate}, \mathsf{mike}\}$.

Classical RBAC allows for great flexibility in layering access control hierarchically in an organization. Indeed, by assigning or de-assigning users to roles, a security officer can immediately grant or revoke authorization to those users for the execution of some specific subset of tasks. Despite all this, RBAC models fail to specify security policies at user level such as *separation of duties (SoD)* and *binding of duties (BoD)*. A SoD is a security policy saying that a subset of tasks must be carried out by different users, whereas BoD says that a subset of tasks must be carried out by the same user. *Authorization constraints* bridge such a gap.

Definition 5 (Authorization constraint). Consider an RBAC model (U, R, T, UA, TA).

- An *atemporal* authorization constraint is a triple (t_i, t_j, A), where $t_i, t_j \in T$ are two non-mutually exclusive tasks, $A \subseteq U \times U$, and $u_i \in users(t_i)$ and $u_j \in users(t_j)$ for each $(u_i, u_j) \in A$. Specifically, A defines the allowed combinations of user assignments to t_i and t_j.

- A *temporal* authorization constraint is a pair $(F, t_i - t_j \in [x, y])$, where F is a conjunction of atoms $t\ \theta\ u$ with $\theta\ \in \{=, \neq\}$, $t, t_i, t_j \in T$ are non-mutually exclusive tasks, $x, y \in \mathbb{R} \cup \{-\infty, +\infty\}$, and $u \in users(t)$.

Our motivation scenario has the following authorization constraints:

- (CardioVisit, ResultAnalysis, {(kate, kate), (mike, mike)})

- (CardioVisit, FinalReport, {(kate, kate), (mike, mike)})

- $(\mathsf{CardioVisit}, \mathsf{BloodTest\text{-}Type1}, \{(\mathsf{kate}, \mathsf{mike}), (\mathsf{mike}, \mathsf{kate})\})$
- $(\mathsf{CardioVisit}, \mathsf{BloodTest\text{-}Type2}, \{(\mathsf{kate}, \mathsf{mike}), (\mathsf{mike}, \mathsf{kate})\})$
- $(\mathsf{BloodTest\text{-}Type2} = \mathsf{mike}, E - S \in [-\infty, 100])$

In Figure 2, atemporal authorization constraints are graphically represented through BPMN natural-language annotations connecting non-mutually exclusive tasks.

Let us now discuss how we can augment the STND of our motivating example to also consider access control. Let (U, R, T, UA, TA) be the RBAC model and $\mathcal{A}_A$ and $\mathcal{A}_T$ be the set of atemporal and temporal authorization constraints discussed so far. Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{DT}, \mathcal{P}, O, L, \mathcal{C} \rangle$ be the STND in Figure 3. The encoding is as follows.

Authorized users. For each $t \in T$ and $u \in users(t)$, we add to $\mathcal{S}$ a decision time point $T_u!$ associated to a Boolean t_u resembling the following interpretation: t is executed by u *if and only if* $t_u = \top$.

In our example we add to $\mathcal{T}$ *the decision time points* $V_m!$, $V_k!$, $R_m!$, $R_k!$, $B1_m!$, $B1_k!$, $B2_m!$, $B2_k!$, $F_m!$, $F_k!$, *and we add to* $\mathcal{P}$ *the Booleans* v_m, v_k, r_m, r_k, $b1_m$, $b1_k$, $b2_m$, $b2_k$, f_m, f_k, *where subscripts* k *and* m *correspond to users* kate *and* mike, *respectively.*

Each task is executed by one authorized user. For each $t \in T$ and $u \in users(t)$, let ℓ be the label of the time points encoding the start and end of t in the STND. Consider any time point in the STND (e.g., S). We add a constraint $(S - S \leq -1, \ell \bigwedge_{u \in users(t)} \neg t_u)$. This constraint imposes that some user will execute t. However, it is not enough. Indeed, it might be the case that more than one user execute the task. For each 2-subset $\{u_i, u_j\} \subseteq users(t)$, we add the constraint $(S - S \leq -1, \ell \wedge t_{u_i} \wedge t_{u_j})$. Note that, despite the number of k subsets of an n-element set is $\binom{n}{k} = \frac{n!}{k!(n-k)!}$, in our case, since k is fixed, we have a number of subsets which is polynomial in the size of $users(t)$. Indeed, we generate exactly $\binom{|users(t)|}{2} = \frac{|users(t)|!}{2!(|users(t)|-2)!} = \frac{|users(t)|(|users(t)|-1)(|users(t)|-2)!}{2(|users(t)|-2)!} = \frac{|users(t)|^2 - |users(t)|}{2}$ constraints.

In our example we add to $\mathcal{C}$ *the constraints* $(S - S \leq -1, \neg v_k \wedge \neg v_m)$, $(S - S \leq -1, \neg r_k \wedge \neg r_m)$, $(S - S \leq -1, b \wedge \neg b1_k \wedge \neg b1_m)$, $(S - S \leq -1, \neg b \wedge \neg b2_k \wedge \neg b2_m)$ *and* $(S - S \leq -1, \neg f_k \wedge \neg f_m)$.

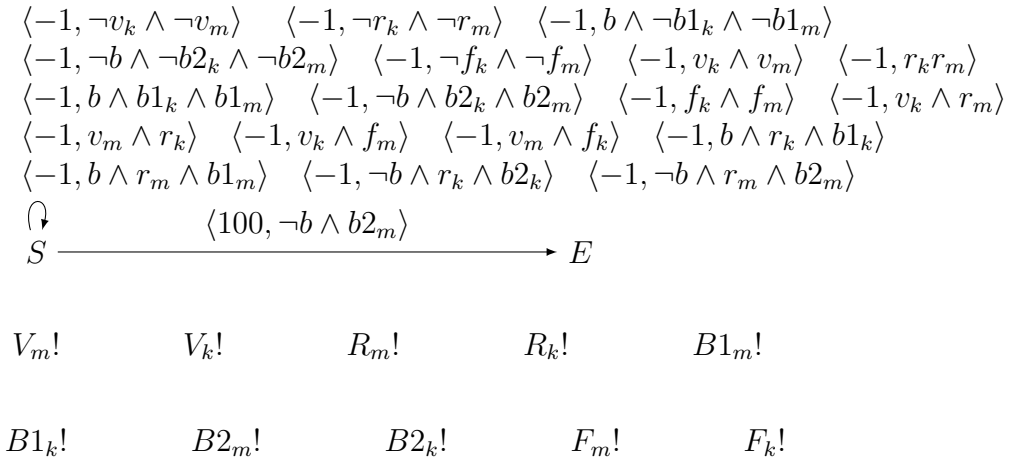


Figure 4: Portion of STND modeling Figure 2 considering access control as well. Note that all decision time points are unconstrained.

Atemporal authorization constraints. For each $(t_i, t_j, A) \in \mathcal{A}_A$, let $\overline{A} = (U \times U) \setminus A$ be the complement of A. Let ℓ_i and ℓ_j be the labels of the time points encoding the start and the end of t_i and t_j in the STND. Consider any time point in the STND (e.g., S). For each $(u_i, u_j) \in \overline{A}$, we add the constraint $(S - S \leq -1, \ell_i \wedge \ell_j \wedge t_{u_i} \wedge t_{u_j})$. These constraints prevent all user assignment combinations which are not allowed.

In our example we add to $\mathcal{C}$ the constraints $(S - S \leq -1, v_k \wedge v_m)$, $(S - S \leq -1, r_k \wedge r_m)$, $(S - S \leq -1, b \wedge b1_k \wedge b1_m)$, $(S - S \leq -1, \neg b \wedge b2_k \wedge b2_m)$, $(S - S \leq -1, f_k)$, $(S - S \leq -1, v_k \wedge r_m)$, $(S - S \leq -1, v_m)$, $(S - S \leq -1, v_k \wedge f_m)$, $(S - S \leq -1, v_m)$, $(S - S \leq -1, b \wedge r_k \wedge b1_k)$ $(S - S \leq -1, b \wedge r_m \wedge b,$ $(S - S \leq -1, \neg b \wedge r_k \wedge b2_k)$, $(S - S \leq -1, \neg b \wedge r_m \wedge b2_m)$.

Temporal authorization constraints. For each $(F, t_i - t_j \in [x, y]) \in \mathcal{A}_T$, let $\ell_F := \bigwedge_{t=u \in F} t_u \bigwedge_{t \neq u \in F} \neg t_u$ be the corresponding STND label encoding F and let ℓ_i and ℓ_j be the labels of the time points encoding the start and the end of t_i and t_j in the STND, respectively. We add the pair of constraints $(t_i - t_j \leq y, \ell_F \wedge \ell_i \wedge \ell_j)$ and $(t_j - t_i \leq -x, \ell_F \wedge \ell_i \wedge \ell_j)$.

In our example we add to $\mathcal{C}$ the (only relevant) constraint $(E - S \leq 100, \neg b \wedge b2_m)$.

Overall, the encoding runs in polynomial time. A graphical representation

of the part of the STND needed to model the discussed access control is given in Figure 4.

6. Experimental Evaluation

We developed KAPPA, a tool for STNDs that takes in input a specification of an STND and acts both as a solver and as a solution verifier. KAPPA relies on SAT4J [29], a Java library compliant with the IPASIR interface that specifies how to interact with a SAT solver [30].

KAPPA implements both `STND-HSCC1` and `STND-HSCC2`. We used an extended implementation of `STND-HSCC1`, which allows for the synthesis of all consistent scenarios as well. In this way, we could carry out a more accurate experimental evaluation comparing the two algorithms when seeking either single or all consistent scenarios.

We first use KAPPA on our case study. Listing 1 shows the specification of the STND in Figure 3a in KAPPA's input language. Each input specification starts by specifying the set of Boolean variables, then it specifies the set of time points by distinguishing normal time points $X \in \mathcal{T} \setminus \mathcal{DT}$, which are specified as `(X:label)`, from decision time points $D! \in \mathcal{DT}$, which are specified as `(D!:boolean:label)`. Finally, each constraint $(Y - X \leq k, \ell)$ is specified as `(Y-X<=k:label)`. We synthesized all possible consistent scenarios and computed all possible early execution schedules of the arising STN-projections. The checking time for this example is negligible. Finally, we verified that all synthesized projections are consistent for the corresponding scenario. Listing 2 shows these results.

After that, we randomly generated 2200 STNDs partitioned in 11 benchmark sets, each one containing 100 consistent STNDs and 100 inconsistent STNDs. Regardless of the set, each STND has exactly 100 time points. The first set (`100TimePoints/10Decisions`) specifies 10 decision time points, the second set (`100TimePoints/11Decisions`) specifies 11 decision time points and so on, up to the eleventh one (`100TimePoints/20Decisions`) that specifies 20 decision time points. Each STND has a maximum number of constraints of $|\mathcal{T}| \times |\mathcal{DT}|$. Time points and constraints are randomly labeled so that the resulting STND is well defined. The weights on labeled edges range from -100 to 100. This data and KAPPA are available online at `https://github.com/matteozavatteri/kappa`.

We ran KAPPA on these benchmark sets without imposing any limit to collect data (time and space) for both `STND-HSCC1` and `STND-HSCC2` when

Listing 1: Specification of Figure 3a in KAPPA.

```
Propositions { a b }

TimePoints {
       (S : ) (VS : ) (VE : ) (A! : a : ) (TS : a) (TE : a) (CS : !a)
       (CE : !a) (JA : ) (RS : ) (RE : ) (B! : b : ) (B1S : b) (B1E : b)
       (B2S : !b) (B2E : !b) (JB : ) (FS : ) (FE : ) (E : )
 }

Constraints {
       (S - VS <= 0 : )
       (VS - S <= 5 : )
       (VS - VE <= -10 : )
       (VE - VS <= 10 : )
       (VE - A <= -1 : )
       (A - VE <= 1 : )
       (A - TS <= -2 : a)
       (TS - A <= 10 : a)
       (TS - TE <= -30 : a)
       (TE - TS <= 30 : a)
       (TE - JA <= -1 : a)
       (JA - TE <= 1 : a)
       (A - CS <= -2 : !a)
       (CS - A <= 20 : !a)
       (CS - CE <= -20 : !a)
       (CE - CS <= 20 : !a)
       (CE - JA <= -1 : !a)
       (JA - CE <= 1 : !a)
       (JA - RS <= -1 : )
       (RS - JA <= 1 : )
       (RS - RE <= -1 : )
       (RE - RS <= 5 : )
       (RE - B <= -10 : )
       (B - RE <= 20 : )
       (B - B1S <= -1 : b)
       (B1S - B <= 10 : b)
       (B1S - B1E <= -20 : b)
       (B1E - B1S <= 30 : b)
       (B1E - JB <= -1 : b)
       (JB - B1E <= 1 : b)
       (B - B2S <= -1 : !b)
       (B2S - B <= 10 : !b)
       (B2S - B2E <= -50 : !b)
       (B2E - B2S <= 120 : !b)
       (B2E - JB <= -1 : !b)
       (JB - B2E <= 1 : !b)
       (JB - FS <= -1 : )
       (FS - JB <= 10 : )
       (FS - FE <= -4 : )
       (FE - FS <= 6 : )
       (FE - E <= -1 : )
       (E - FE <= 1 : )
       (VS - RE <= -35 : )
       (RE - VS <= 45 : )
       (S - E <= -90 : )
       (E - S <= 156 : )
}
```

Listing 2: Validation and verification of the STND in Figure 3a with KAPPA.

```
$ java -jar kappa.jar Fig1a.stnd --hscc2-all Fig1a.s
hscc2
all = true
Consistent
Saving schedules for 2 scenario(s) to Fig1a.s

$ java -jar kappa.jar Fig1a.stnd --verify Fig1a.s
Scenario: !a b
Schedule: {Z = 0, S = 0, VS = 0, VE = 10, A = 11, CS = 13, CE = 33, JA = 34, RS = 35, RE =
    36, B = 46, B1S = 47, B1E = 72, JB = 73, FS = 83, FE = 89, E = 90}
Scenario: !a !b
Schedule: {Z = 0, S = 0, VS = 0, VE = 10, A = 11, CS = 13, CE = 33, JA = 34, RS = 35, RE =
    36, B = 46, B2S = 47, B2E = 97, JB = 98, FS = 99, FE = 103, E = 104}
SAT!
```

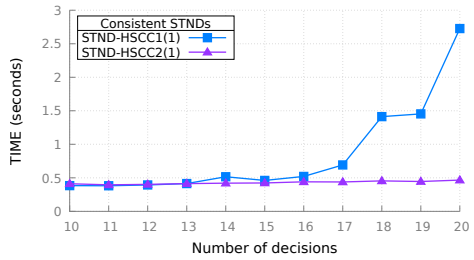
seeking either a single or all consistent scenarios.

We graphically show the results in Figure 5, where x-axes always represent the number (#) of decision time points (i.e., the set under analysis) and y-axes represent either the average time elapsed or space consumed when analyzing the instances in that set.

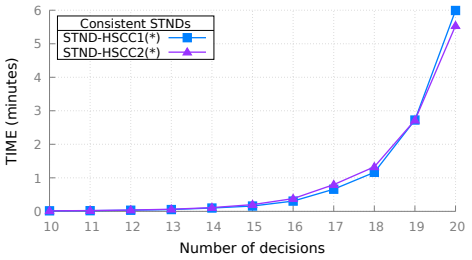
Figure 5a shows the results of the analysis run on the sets containing consistent STNDs when seeking a single consistent scenario. The graph shows that `STND-HSCC2` is significantly faster than `STND-HSCC1` for STNDs specifying more than 16 decision time points. Figure 5b shows the results of the same analysis when seeking all consistent scenarios. Despite a normal general worsening of performances (all consistent scenarios are sought and not just one) `STND-HSCC2` is faster than `STND-HSCC1` for STNDs specifying 20 decisions. Figure 5c shows the results of the analysis on the sets containing inconsistent STNDs. `STND-HSCC2` has no competitors here and its consistency checking times are significantly lower then those of `STND-HSCC1`, whereas `STND-HSCC1` starts also having a serious exponential blow up for STNDs specifying more than 14 decisions. Figure 5d shows the average space consumed when synthesizing all consistent scenarios. The curve grows exponentially according to the number of decision time points (recall that `STND-HSCC1` and `STND-HSCC2` return the same set of consistent scenarios in such an analysis, therefore we only show the data for `STND-HSCC2`).

We verified all synthesized solutions. No constraint was violated.

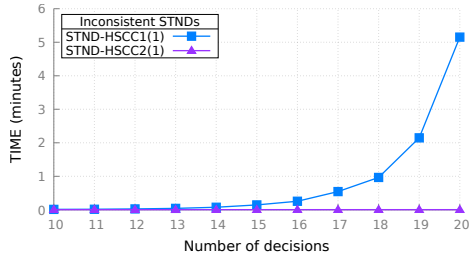
Finally, Listing 3 shows how to extend Listing 1, to consider access control, whereas Listing 4 shows the validation and execution with KAPPA. The synthesis of all scenarios also corresponds to synthesizing all possible valid



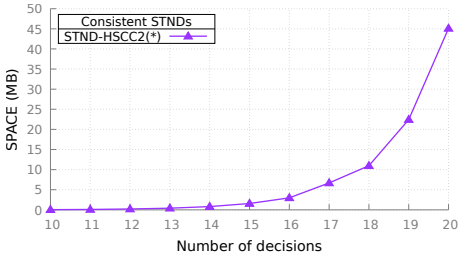
(a) Consistency checking time on consistent STNDs when seeking a single consistent scenario.



(b) Consistency checking time on consistent STNDs when seeking all consistent scenarios.



(c) Consistency checking time on inconsistent STNDs.



(d) Space consumed when synthesizing all consistent scenarios.

Figure 5: Experimental evaluation with KAPPA.

combinations of user assignments. We still can go through two process paths ($\neg a \wedge \neg b$ and $\neg a \wedge b$). In the former the unique user assignment is to employ kate as a cardiologist and mike as a pathologist (since this process path undergoes BloodTest-Type1), whereas in the latter these two users can interchange (since this process path undergoes BloodTest-Type2).

7. Translating STNDs to DTNs and Back

In this section we will consider the mapping from STNDs to DTNs and vice versa. DTNs are expressed through another constraint-based formalism allowing for the representation of disjoint temporal constraints [21]. STNDs and DTNs have some similarities in allowing one to specify multiple possible plans, where a specific subset of the given temporal constraints will hold.

Consistency of STNDs and consistency of DTNs are both NP-complete [1,

Listing 3: Extension to the specification of Listing 1 to consider access control in KAPPA.

```
Propositions {
...
vk vm
rk rm
b1k b1m
b2k b2m
fk fm
}

TimePoints {
...
(Vk! : vk : )
(Vm! : vm : )
(Rk! : rk : )
(Rm! : rm : )
(B1k! : b1k : b)
(B1m! : b1m : b)
(B2k! : b2k : !b)
(B2m! : b2m : !b)
(Fk! : fk : )
(Fm! : fm : )
}

Constraints {
...
(S - S <= -1 : !vk !vm)
(S - S <= -1 : !rk !rm)
(S - S <= -1 : b !b1k !b1m)
(S - S <= -1 : !b !b2k !b2m)
(S - S <= -1 : !fk !fm)
(S - S <= -1 : vk vm)
(S - S <= -1 : rk rm)
(S - S <= -1 : b b1k b1m)
(S - S <= -1 : !b b2k b2m)
(S - S <= -1 : fk fm)
(S - S <= -1 : vk rm)
(S - S <= -1 : vm rk)
(S - S <= -1 : vk fm)
(S - S <= -1 : vm fk)
(S - S <= -1 : b rk b1k)
(S - S <= -1 : b rm b1m)
(S - S <= -1 : !b rk b2k)
(S - S <= -1 : !b rm b2m)
(E - S <= 100 : !b b2m)
}
```

Listing 4: Validation and verification with KAPPA of the STND encoded in Listing 1 and extended as in Listing 3.

```
$ java -jar kappa.jar Fig1a.ac.stnd --hscc2-all Fig1a.ac.s
hscc2
all = true
Consistent
Saving schedules for 3 scenario(s) to Fig1a.ac.s

$ java -jar kappa.jar Fig1a.ac.stnd --verify Fig1a.ac.s
Scenario: !a !b !vk vm !rk rm b2k !b2m !fk fm
Schedule: {Z = 0, S = 0, VS = 0, VE = 10, A = 11, CS = 13, CE = 33, JA = 34, RS = 35, RE =
     36, B = 46, B2S = 47, B2E = 97, JB = 98, FS = 99, FE = 103, E = 104, Vk = 0, Vm = 0, Rk
      = 0, Rm = 0, B2k = 46, B2m = 46, Fk = 0, Fm = 0}

Scenario: !a b vk !vm rk !rm !b1k b1m fk !fm
Schedule: {Z = 0, S = 0, VS = 0, VE = 10, A = 11, CS = 13, CE = 33, JA = 34, RS = 35, RE =
     36, B = 46, B1S = 47, B1E = 72, JB = 73, FS = 83, FE = 89, E = 90, Vk = 0, Vm = 0, Rk =
      0, Rm = 0, B1k = 46, B1m = 46, Fk = 0, Fm = 0}

Scenario: !a b !vk vm !rk rm b1k !b1m !fk fm
Schedule: {Z = 0, S = 0, VS = 0, VE = 10, A = 11, CS = 13, CE = 33, JA = 34, RS = 35, RE =
     36, B = 46, B1S = 47, B1E = 72, JB = 73, FS = 83, FE = 89, E = 90, Vk = 0, Vm = 0, Rk =
      0, Rm = 0, B1k = 46, B1m = 46, Fk = 0, Fm = 0}
SAT!
```

18, 21]. As a result, there exist polynomial-time reductions to translate any STND into a DTN and vice versa. It is thus interesting, from the modeling point of view, to study how (and how difficult it is) to move from one formalism to the other one, and vice versa, when representing real-world temporal constraints.

DTNs are temporal networks that allow for disjunctions of temporal constraints (i.e., alternatives) in a temporal problem and are a possible formalism to model the *disjunctive temporal problem* (DTP). For example, we might want that once an event modeled by a time point X happened, another event modeled by a time point Y happens *either* after 10 (seconds, minutes, hours, ...) *or* within 5. Such a constraint would look like

$$(X - Y \leq -10) \vee (Y - X \leq 5)$$

Any assignment of real values to X and Y satisfies the constraint if it satisfies (at least) one disjunct.

Differently from the initial proposal in [1], where disjunctions of intervals were allowed on the *same pairs of time points* only, the work we consider here is the one in [21], not having such a restriction.

Definition 6. A *Disjunctive Temporal Network* (*DTN*) is a pair $\langle \mathcal{T}, \mathcal{C} \rangle$, where

- $\mathcal{T}$ is the usual finite set of time points, and
- $\mathcal{C}$ is a finite set of temporal constraints each one having the form

$$\underbrace{(Y_1 - X_1 \leq k_1) \vee \cdots \vee (Y_n - X_n \leq k_n)}_{n \text{ disjuncts (atoms)}}$$

  where $X_i, Y_i \in \mathcal{T}$ and $k_i \in \mathbb{R}$. A temporal constraint is *non-disjunctive* if and only if it contains one disjunct, *disjunctive* otherwise.

  A DTN is *consistent* if there exists an assignment of real values to all time points (i) always satisfying all non-disjunctive constraints and (ii) satisfying at least one disjunct for each disjunctive constraint.

We write $D(i)$ to shorten the i^{th} disjunctive constraint and more specifically $D(i, j)$ to refer to the j^{th} disjunct of the i^{th} disjunctive constraint [21].

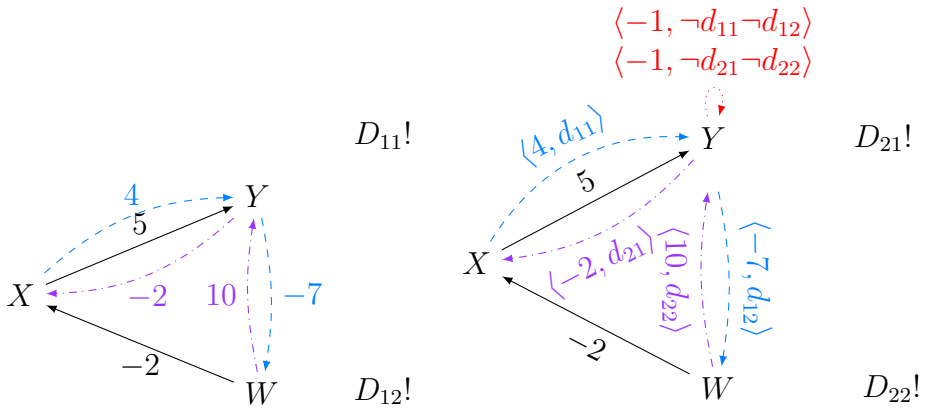
We represent a DTN graphically through a *colored multi graph*, where *red edges* model non-disjunctive constraints (i.e., those constraints that must always hold), whereas *colored edges* (different from red) model disjunctive constraints (i.e., those $D(i)$s for which at least one disjunct must hold). Each disjunctive constraint is assigned to a different color (we also use a unique line pattern for each color).

To give an example, consider the following DTN, whose corresponding colored multi graph is shown in Figure 6a.

- $\mathcal{T} = \{X, Y, W\}$
- $\mathcal{C} = \{\overbrace{(Y - X \leq 5), (X - W \leq -2)}^{\text{must always hold}}, \overbrace{\overbrace{(Y - X \leq 4)}^{D(1,1)} \vee \overbrace{(W - Y \leq -7)}^{D(1,2)}}^{D(1)},$
  $\underbrace{\underbrace{(X - Y \leq -2)}_{D(2,1)} \vee \underbrace{(Y - W \leq 10)}_{D(2,2)}}_{D(2)}\}$

The DTN in Figure 6a is consistent, and, for example, the assignment $X = 0$, $Y = 3$ and $W = 5$ satisfies:

- all non-disjunctive constraints (solid black edges),



(a) Example of DTN. (b) Corresponding STND.

Figure 6: Representing and encoding DTNs into STNDs.

- $D(1,1)$ but not $D(1,2)$ for the disjunctive constraints $D(1)$ (dashed blue edges),
- $D(2,1)$ and also $D(2,2)$ for the disjunctive constraint $D(2)$ (dashdotted purple edges).

We now proceed by specifying and describing the reductions. We first give a strongly polynomial-time reduction from DTNs to STNDs and then the vice versa reduction (and we provide examples throughout this discussion).

7.1. Reducing DTNs to STNDs

We reduce the DTN in Figure 6a to the corresponding STND in Figure 6b as follows.

We generate a "core" STND containing all time points and all non-disjunctive constraints of the starting DTN without labeling them since all time points must always be assigned a value and all non-disjunctive constraints must always be satisfied.

For each disjunctive constraint $D(i)$ in the DTN, we add to the STND as many decision time points $D_{ij}!$ as the number of disjuncts $D(i,j)$. These decision time points are not constrained to any other time point in the STND (i.e., free to take any value). Any disjunct $D(i,j)$ in the DTN appears as a constraint in the STND labeled by d_{ij} (the proposition associated to $D_{ij}!$) so that when $d_{ij} = \top$, the disjunct of the DTN (labeled constraint in the STND)

must hold and when $d_{ij} = \bot$ we are not obliged to satisfy it. Moreover, we impose that at least one disjunct $D(i, j)$ for any disjunctive constraint $D(i)$ must hold (otherwise, it would be possible to disable them all by setting all d_{ij} to $\bot$). We enforce this condition by adding a negative self loop labeled by $\neg d_{ij_1} \dots \neg d_{ij_n}$ on any time point of the STND.

In Figure 6b, we added four decision time points $D_{11}!$, $D_{12}!$, $D_{21}!$ and $D_{22}!$ and the constraints $X \to Y$ labeled by $\langle 4, d_{11} \rangle$, $Y \to W$ labeled by $\langle -7, d_{12} \rangle$, $Y \to X$ labeled by $\langle -2, d_{21} \rangle$ and $W \to Y$ labeled by $\langle 10, d_{22} \rangle$ to switch on and off the disjuncts $D(1, 1)$, $D(1, 2)$, $D(2, 1)$ and $D(2, 2)$ through the truth value assignments to d_{11}, d_{12}, d_{21} and d_{22}. Finally, we added two negative self loops $Y \to Y$ labeled by $\langle -1, \neg d_{11} \neg d_{12} \rangle$ and $\langle -1, \neg d_{21} \neg d_{22} \rangle$ to prevent a disjunctive constraint $D(i)$ from being excluded (red self loop at Y). Note that the "-1" is meaningless: *any negative number* (e.g., -3, -159 or $-\epsilon$) is fine for this purpose. Likewise, the choice of time point Y is meaningless too. Any time point would be fine for this purpose (e.g., $X \to X$ labeled by the same constraints). Negative self loops are the more intuitive way to enforce these conditions. However, nothing would have prevented us from creating cycles of negative sum with respect to these labels involving many time points.

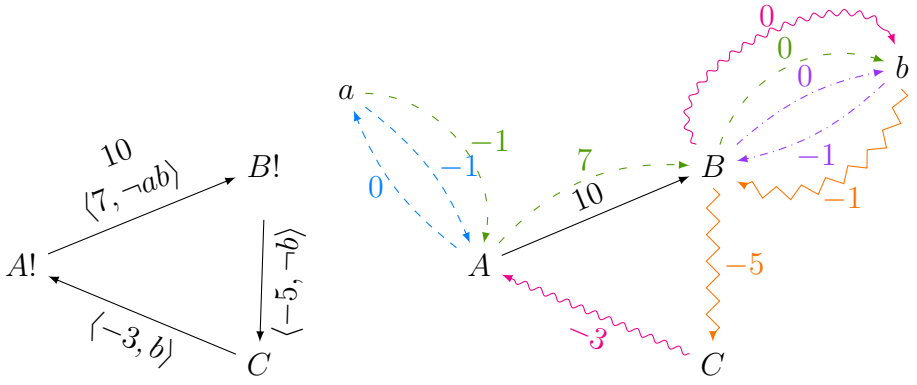
To ease reading and understand "what goes where", we colored the STND in Figure 6b with the same colors of the DTN in Figure 6a and showed the added negative cycles in red.

This encoding is strongly polynomial. The number of time points in the STND is equal to the number of time points in the DTN plus as many decision time points as the number of disjuncts $D(i, j)$ contained in all disjunctive constraints $D(i)$ in the DTN. The number of constraints in the STND is equal to the the number of non-disjunctive constraints plus as many constraints as the number of disjuncts $D(i, j)$ contained in all disjunctive constraints $D(i)$ in the DTN plus as many constraints as the number of disjunctive constraints $D(i)$ to model negative loops.

Any consistent scenario in the STND says which disjuncts (at least one for each disjunctive constraint) are satisfied for the solution. If the STND is inconsistent, so is the DTN.

7.2. Reducing STNDs to DTNs

We reduce the STND in Figure 7a to the corresponding STND in Figure 7b as follows. First of all, if the STND has labels on nodes we convert it



(a) Example of (streamlined) STND.

(b) Corresponding DTN.

Figure 7: Encoding STNDs into DTNs.

to its streamlined version having only label on edges. The process of streamlining a temporal network was first discussed in [31] for CSTNs. However, that process works for STNDs as well (as consistency is a special case of controllability in which no uncontrollable part is considered). Then, we generate a "core" DTN having the same set of time points of the STND (we drop all "!" from the names). Also, all unlabeled constraints in the STND become non-disjunctive constraints in the DTN.

For each proposition d associated to a decision time point $D!$ in the STND, we add to the DTN a time point d and the disjunctive constraint

$$\underbrace{(d - D \leq 0)}_{\text{means } d = \bot} \vee \underbrace{(D - d \leq -1)}_{\text{means } d = \top}$$

where the former says that d "occurs within" D, whereas the latter says that d occurs at least 1 after D (a way to to simulate a Boolean condition).

Now, every constraint $X \to Y$ labeled by $\langle k, d\neg ef \dots \rangle$ (in the STND) implies the following "meta constraint" in the DTN:

$$(\underbrace{D - d \leq -1}_{d} \wedge \underbrace{e - E \leq 0}_{\neg e} \wedge \underbrace{F - f \leq -1}_{f} \dots) \Rightarrow Y - X \leq k$$

which can be rewritten as

$$\neg(D - d \leq -1 \wedge e - E \leq 0 \wedge F - f \leq -1 \dots) \vee Y - X \leq k$$

and finally simplified to

$$\underbrace{(d - D \leq 0)}_{\neg d} \vee \underbrace{(E - e \leq -1)}_{e} \vee \underbrace{(f - F \leq 0)}_{\neg f} \cdots \vee (Y - X \leq k)$$

Note that for any D and d (in the DTN) we define $\neg(D - d \leq -1)$ as $d - D \leq 0$ and $\neg(d - D \leq 0)$ as $D - d \leq -1$ since they are abstracting Boolean conditions only and we are not therefore interested in a specific numeric value. Therefore, for any labeled constraint in the STND we add such a disjunctive constraint to the DTN.

In Figure 7b we add two time points a and b and the following constraints:

- $B - A \leq 10$ (solid black edges),
- $D(1)\colon (a - A \leq 0) \vee (A - a \leq -1)$ (dashed blue edges),
- $D(2)\colon (b - B \leq 0) \vee (B - b \leq -1)$ (dashdotted purple edges),
- $D(3)\colon (A - a \leq -1) \vee (b - B \leq 0) \vee (B - A \leq 7)$ (loosely dashed green edges),
- $D(4)\colon (B - b \leq -1) \vee (C - B \leq -5)$ (zigzag orange edges),
- $D(5)\colon (b - B \leq 0) \vee (A - C \leq -3)$ (snake magenta edges).

We show the "colored" DTN graph in Figure 7b. Now, the DTN is consistent if and only if the STND is so. A solution of the DTN corresponds to a consistent scenario in the STND. The truth value assignment to the propositions in the STND depends on the real value assignments to the time points modeling those propositions in the DTN. For any proposition d in the STND, d is false iff in the DTN the time point d has a value not greater than $D!$ and d is true in the STND iff in the DTN the value of time point d is greater than $D!$ (the assignment to the other time points defines a schedule consistent for the scenario).

This encoding is strongly polynomial. The number of time points in the DTN is the same of that in the STND plus as many time points as the number of propositions in the STND. The number of constraints in the DTN is given by the number of unlabeled constraints in the STND (non-disjunctive constraints in the DTN), plus as many disjunctive constraints as the number of labeled constraints in the STND (i.e., whose labels are different from $\boxdot$).

Also, for any disjunctive constraint $D(i)$ in the DTN, the number of disjuncts of $D(i)$ is $n+1$ where n is the number of literals contained in the label of the corresponding constraint in the STND and the "+1" refers to the inequality.

8. Related Work

Let us now consider the main contributions that faced the issue of specifying/checking temporal constraints, while taking into consideration also conditions and decisions.

Drake [16] is an executive for temporal plans with choices. Such plans are represented as Labeled STNs. In such extended STNs, constraints are labeled with environments (set of instantiated discrete variables). Decision points are not explicitly represented and time points are unlabeled. During the execution, choices are discriminated by generating conflicts with respect to the moment Drake decides to schedule some event. With respect to Drake, STNDs rely on a more structured approach by using labels instead of environments. Indeed, labeled time points are used to prevent their execution when some literal in the label is still unknown. And it enforces well-defined properties, as making decisions only upon the execution of the related decision time points.

A *Disjunctive Temporal Network (DTN)* [21] extends an STN with disjunctive constraints. Any solution to a DTN must satisfy all non-disjunctive constraints (i.e., STN-constraints) and at least one disjunct for each disjunctive constraint. Labeled STNs and DTNs are equivalent [16]. DTNs and STNDs are equivalent too, therefore, Labeled STNs are equivalent to STNDs as well.

Temporal Plan Networks (TPNs) [17] extend STNs by adding decision nodes and symbolic constraints to model temporal plans with controllable choices modeled as outgoing edges from a decision node. Each outgoing edge corresponds to a specific decision. Time points are not labeled and activities are modeled as pair of non-decision nodes (start,end). A symbolic constraint is either Ask(c) (is c true?) or Tell(c) (c is true!), where c a literal. Symbolic constraints allow for the exclusion of some activities from a given execution of a plan. A plan is consistent if it satisfies both temporal and symbolic constraints. TPNs do not specify more than one temporal constraint on the same edge. Consistency is checked by visiting the nodes of the graph from start to end taking one edge (modeling a decision) at a time. If the resulting

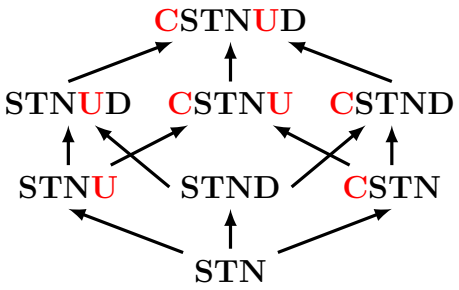


Figure 8: A hierarchy of simple temporal networks. Acronyms containing **D** (resp., **C**) mean that formalisms deal with controllable (resp., uncontrollable) conditionals, whereas those containing **U**, mean that formalisms deal with uncontrollable durations. We highlight uncontrollable parts in red.

STN-projection is inconsistent, the algorithm backtracks to the last decision node that still has unexplored outgoing edges.

Both controllable and uncontrollable choices (i.e., decisions and conditions) are modeled in *Pike* [32], which is an executive for *Temporal Plan Networks with Uncertainty (TPNUs)*, which extend TPNs with uncontrollable choices. Pike adapts its controllable choices to the uncontrollable ones made by a human. STNDs do not have uncontrollable choices. A *Controllable Conditional Temporal Problem (CCTP)* [28] is an optimization problem for temporal plans with conditions and thus it cannot be directly compared with STNDs. *CCTPs with Uncertainty (CCTPUs)* [33] address temporal plans with controllable choices and uncontrollable durations, whereas in [34], TPNUs are extended to support uncontrollable durations (strong controllability only). In both works, relaxation techniques are used to restore controllability of an uncontrollable plan. To this regard, in this paper we focused on STNDs that do not have uncontrollable parts.

Several extensions of *STNs* address uncertainties for both temporal constraints and different/alternative execution paths. For example, *Simple Temporal Networks with Uncertainty* (*STNUs*, [3]) add uncontrollable (but bounded) durations represented through contingent links, whereas *Conditional Simple Temporal Networks* (*CSTNs*, [26]) and the former *Conditional Temporal Problem* (*CTP*, [35]) extend STNs by considering conditional constraints depending on uncontrollable truth value assignments associated to the occurrence of some special kind of time points called *observations*. Finally, *Conditional Simple Temporal Networks with Uncertainty* (*CSTNUs*, [36, 4]) merge STNUs and CSTNs, whereas *Conditional Simple Temporal Networks*

with Uncertainty and Decisions (*CSTNUDs*, [19, 25, 37]) add conditional constraints with controllable truth value assignments made when executing some special time points called *decisions*. CSTNUDs encompass all previous formalisms. All these networks extend STNs by adding at least an uncontrollable part. As we already underlined, here we do not address any uncontrollable part. STNDs derive from [18] by removing uncontrollable conditionals and from [19, 25] by removing uncontrollable conditionals and uncontrollable durations. This work extends [18] by providing `STND-HSCC2` both to speed up the consistency checking phase and to allow for the synthesis of all consistent scenarios. This work is, however, incomparable with the proposals in [19, 25], as that contributions employ timed game automata. Figure 8 provides a hierarchy of simple temporal networks.

As for access-control networks, there also have been some attempts to consider time and resources together, e.g., *Access Controlled Temporal Networks* (*ACTNs*, [38]) and *Conditional Simple Temporal Networks with Uncertainty and Resources* (*CSTNURs*, [39]), which were preceded by an initial proposal in [40]. However, neither ACTNs nor CSTNURs employ decision time points. Research on temporal networks has inspired a recent line of work in which controllability analysis focused on resource allocation under uncertainty employing a qualitative temporal approach instead of a quantitative one. This is the case of access controlled workflows (i.e., business processes) investigated in [11, 41] and of extensions of constraint networks proposed in [42, 43], where *Constraint Networks Under Conditional Uncertainty* (*CNCUs*) are introduced (see also [44, 45]). In these proposals, temporal relations are only qualitative (specifically, "before/after") and decision time points are not considered. A short summary of temporal and resource controllability based on constraint networks and considered either in isolation or simultaneously can be found in [46].

Planning as satisfiability was formally introduced in [47, 48] and relies on a set of axioms where any model corresponds to a valid plan. Before that, planning was based on deduction. Recently, more performant SAT encodings have been provided (e.g., in [49]). However, none of these approaches is incremental and thus they are incomparable with ours.

Gocht and Balyo [50] provide an incremental SAT solving approach for SAT-based planning and prove that incremental SAT solving outperforms the non-incremental one, but they do not address temporal constraints. Our work does not model "transitions" but applies shortest path algorithms, incrementally, on STN-projections.

Temporal induction is an incremental technique to check safety properties on finite state machines and it is strongly related to bounded model checking [51]. It is similar to SAT-based planning and allows for the detection of the unreachability of a goal. Our analysis is not bounded with respect to the "depth".

Satisfiability modulo theory (*SMT*, [52]) can describe STNDs by using a fragment of Linear Real Arithmetic called Difference Logic. However, SMT-solvers do not guarantee to find early schedules. A run of an HSCC algorithm and a run of an SMT-solver are not guaranteed to return the same consistent scenarios (Boolean part). A fairer comparison is when both HSCC-algorithms and SMT-solvers look for all consistent scenarios, but then we should make sure that the SMT-solver does not return more than one schedule for each consistent scenario.

Incremental task planning adopts incremental features of SMT-solvers to extend a constraint-based task planning to motion domains [53]. As it adopts a probability-based approach and focus on the motion domain, it cannot be directly compared to the work we propose in this paper.

9. Conclusions and Future Work

In this paper we extended STNDs to model and verify business processes having temporal and access-control constraints (often intertwined) and discussed a real-world case study from the domain of healthcare processes. In particular, we proposed `STND-HSCC2`, a novel hybrid SAT-based consistency checking algorithm for STNDs. This new version of the algorithm still relies on a SAT solver but differently from the previous one, it exploits partial truth value assignments to hunt down negative cycles in STN-projections as early as possible. The previous algorithm tested STN-projections for negative cycles by iterating on *complete* models returned by the SAT solver. When the SAT solver makes an assumption, we project the STND over the current truth value assignment of the propositions (i.e., partial model) extended with this new assumption. If the projected STN is inconsistent, we add a clause to the SAT solver to exclude that scenario, else we let the solver go. We then discussed how to represent access-control constraints in a STDN. We implemented our approach and provided Kappa, a tool for STNDs that implements `STND-HSCC1` and `STND-HSCC2` both supporting the synthesis of

single or all consistent scenarios[1] and we compared the results. The more inconsistent STN-projections an STND admits, the better `STND-HSCC2` performs. The saved solutions allow for an offline planning in which all decisions are made before starting and all time points have already been scheduled to execute as soon as possible. Finally, we showed how to translate any STND to an equivalent DTN and vice versa. Considering this equivalence result, one could wonder whether to use STNDs or DTNs for real world temporal business processes. Many different perspectives have to be considered for that. Indeed, on one hand, STNDs allow for an easier modeling of temporal business processes with controllable conditional paths, by exploiting labels one edges and nodes. Thus, the resulting mapping would be more "readable" and simpler to be analyzed for debugging and fine tuning. On the other hand, an experimental analysis on different kinds of such processes would be required to discover whether there are differences in the performances of algorithms checking such networks, according to their specific features (e.g., number of decisions, number of labeled constraints, and so on).

As future work, we plan to investigate optimizations to reduce the size of the CNF clauses added on the fly. We also plan to give a metric suggesting the best algorithm to use depending on the form of the STND in input.

Acknowledgments

This work was partially supported by MIUR, Project *Italian Outstanding Departments, 2018-2022*, and by INdAM, GNCS 2020, Project *Strategic Reasoning and Automated Synthesis of Multi-Agent Systems*.

References

[1] R. Dechter, I. Meiri, J. Pearl, Temporal constraint networks, Artificial Intelligence 49 (1-3) (1991) 61–95.

[2] L. Chittaro, A. Montanari, Temporal representation and reasoning in artificial intelligence: Issues and approaches, Ann. Math. Artif. Intell. 28 (1-4) (2000) 47–106.

[1] Another minor contribution is the extension of `STND-HSCC1` in our tool to support the synthesis of all consistent scenarios.

[3] P. H. Morris, N. Muscettola, T. Vidal, Dynamic control of plans with temporal uncertainty, in: 17th International Joint Conference on Artificial Intelligence, IJCAI 2001, Morgan Kaufmann, 2001, pp. 494–502.

[4] L. Hunsberger, R. Posenato, Sound-and-Complete Algorithms for Checking the Dynamic Controllability of Conditional Simple Temporal Networks with Uncertainty, in: 25th International Symposium on Temporal Representation and Reasoning (TIME 2018), Vol. 120 of Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, pp. 14:1–14:17. doi:10.4230/LIPIcs.TIME.2018.14.

[5] L. Anselma, P. Terenziani, S. Montani, A. Bottrighi, Towards a comprehensive treatment of repetitions, periodicity and temporal constraints in clinical guidelines, Artificial Intelligence in Medicine 38 (2) (2006) 171–195. doi:10.1016/j.artmed.2006.03.007.
URL https://doi.org/10.1016/j.artmed.2006.03.007

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

[7] C. Combi, M. Gozzi, R. Posenato, G. Pozzi, Conceptual modeling of flexible temporal workflows, TAAS 7 (2) (2012) 19:1–19:29. doi:10.1145/2240166.2240169.
URL https://doi.org/10.1145/2240166.2240169

[8] A. Lanz, M. Reichert, B. Weber, Process time patterns: A formal foundation, Inf. Syst. 57 (2016) 38–68. doi:10.1016/j.is.2015.10.002.
URL https://doi.org/10.1016/j.is.2015.10.002

[9] R. Posenato, A. Lanz, C. Combi, M. Reichert, Managing time-awareness in modularized processes, Software and Systems Modeling 18 (2) (2019) 1135–1154. doi:10.1007/s10270-017-0643-4.
URL https://doi.org/10.1007/s10270-017-0643-4

[10] E. Bertino, RBAC models - concepts and trends, Comput. Secur. 22 (6) (2003) 511–514.

[11] M. Zavatteri, C. Combi, R. Posenato, L. Viganò, Weak, strong and dynamic controllability of access-controlled workflows under conditional uncertainty, in: Business Process Management - 15th International Conference, BPM 2017, Springer, 2017, pp. 235–251. doi:10.1007/978-3-319-65000-5_14.

[12] S. Biffl, B. Thurnher, G. Goluch, D. Winkler, W. Aigner, S. Miksch, An empirical investigation on the visualization of temporal uncertainties in software engineering project planning, in: ISESE, IEEE Computer Society, 2005, pp. 437–446.

[13] C. Flores, M. Sepúlveda, Temporal specification of business processes through project planning tools, in: Business Process Management Workshops, Vol. 66 of Lecture Notes in Business Information Processing, Springer, 2010, pp. 85–96.

[14] J. Burgelman, M. Vanhoucke, Maximising the weighted number of activity execution modes in project planning, Eur. J. Oper. Res. 270 (3) (2018) 999–1013.

[15] S. Gueorguiev, M. Harman, G. Antoniol, Software project planning for robustness and completion time in the presence of uncertainty using multi objective search based software engineering, in: GECCO, ACM, 2009, pp. 1673–1680.

[16] P. R. Conrad, B. C. Williams, Drake: An efficient executive for temporal plans with choice, Journal of Artificial Intelligence Research 42 (1) (2011) 607–659.

[17] P. Kim, B. C. Williams, M. Abramson, Executing reactive, model-based programs through graph-based temporal planning, in: IJCAI 2001, Morgan Kaufmann, 2001, pp. 487–493.

[18] M. Cairo, C. Combi, C. Comin, L. Hunsberger, R. Posenato, R. Rizzi, M. Zavatteri, Incorporating decision nodes into conditional simple temporal networks, in: 24th International Symposium on Temporal Representation and Reasoning (TIME 2017), Vol. 90 of LIPIcs, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017, pp. 9:1–9:17. doi:10.4230/LIPIcs.TIME.2017.9.

[19] M. Zavatteri, Conditional simple temporal networks with uncertainty and decisions, in: 24th International Symposium on Temporal Representation and Reasoning (TIME 2017), Vol. 90 of LIPIcs, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017, pp. 23:1–23:17. doi:10.4230/LIPIcs.TIME.2017.23.

[20] M. Zavatteri, C. Combi, R. Rizzi, L. Viganò, Hybrid sat-based consistency checking algorithms for simple temporal networks with decisions, in: J. Gamper, S. Pinchinat, G. Sciavicco (Eds.), 26th International Symposium on Temporal Representation and Reasoning, TIME 2019, October 16-19, 2019, Málaga, Spain, Vol. 147 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 16:1–16:17. doi:10.4230/LIPIcs.TIME.2019.16.
URL https://doi.org/10.4230/LIPIcs.TIME.2019.16

[21] K. Stergiou, M. Koubarakis, Backtracking algorithms for disjunctions of temporal constraints, Artificial Intelligence 120 (1) (2000) 81–117.

[22] C. L. Lafortuna, F. Agosti, R. Galli, C. Busti, S. Lazzer, A. Sartorio, The energetic and cardiovascular response to treadmill walking and cycle ergometer exercise in obese women, European Journal of Applied Physiology 103 (6) (2008) 707. doi:10.1007/s00421-008-0758-y.
URL https://doi.org/10.1007/s00421-008-0758-y

[23] Object Management Group, Business Process Model and Notation (BPMN), v2.0.2, http://www.omg.org/spec/BPMN/2.0.2/.

[24] J. Joshi, E. Bertino, U. Latif, A. Ghafoor, A generalized temporal role-based access control model, IEEE Trans. Knowl. Data Eng. 17 (1) (2005) 4–23.

[25] M. Zavatteri, L. Viganò, Conditional simple temporal networks with uncertainty and decisions, Theoretical Computer Science 797 (2019) 77–101. doi:10.1016/j.tcs.2018.09.023.

[26] L. Hunsberger, R. Posenato, C. Combi, A Sound-and-Complete Propagation-based Algorithm for Checking the Dynamic Consistency of Conditional Simple Temporal Networks, in: 22nd International Symposium on Temporal Representation and Reasoning (TIME 2015), IEEE, 2015, pp. 4–18. doi:10.1109/TIME.2015.26.

[27] H. Li, B. Williams, Generalized conflict learning for hybrid discrete/linear optimization, in: CP 2005, Springer, 2005, pp. 415–429.

[28] P. Yu, B. C. Williams, Continuously relaxing over-constrained conditional temporal problems through generalized conflict learning and resolution, in: IJCAI 2013, IJCAI/AAAI, 2013, pp. 2429–2436.

[29] D. L. Berre, A. Parrain, The sat4j library, release 2.2, JSAT 7 (2-3) (2010) 59–6.
URL http://jsat.ewi.tudelft.nl/content/volume7/JSAT7_4_LeBerre.pdf

[30] T. Balyo, A. Biere, M. Iser, C. Sinz, SAT race 2015, Artificial Intelligence 241 (2016) 45–65.

[31] M. Cairo, L. Hunsberger, R. Posenato, R. Rizzi, A streamlined model of conditional simple temporal networks - semantics and equivalence results, in: 24th International Symposium on Temporal Representation and Reasoning, TIME 2017, Vol. 90 of LIPIcs, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017, pp. 10:1–10:19. doi:10.4230/LIPIcs.TIME.2017.10.

[32] S. J. Levine, B. C. Williams, Concurrent plan recognition and execution for human-robot teams, in: ICAPS 2014, AAAI Press, 2014.

[33] P. Yu, C. Fang, B. C. Williams, Resolving uncontrollable conditional temporal problems using continuous relaxations, in: ICAPS 2014, AAAI Press, 2014.

[34] E. Karpas, S. J. Levine, P. Yu, B. C. Williams, Robust execution of plans for human-robot teams, in: ICAPS 2015, AAAI Press, 2015, pp. 342–346.

[35] I. Tsamardinos, T. Vidal, M. E. Pollack, CTP: A new constraint-based formalism for conditional, temporal planning, Constraints 8 (4) (2003) 365–388.

[36] L. Hunsberger, R. Posenato, C. Combi, The Dynamic Controllability of Conditional STNs with Uncertainty, in: Workshop on Planning and Plan Execution for Real-World Systems (PlanEx) at ICAPS-2012, 2012,

pp. 1–8.
URL http://arxiv.org/abs/1212.2005

[37] M. Zavatteri, R. Rizzi, T. Villa, Strong controllability of temporal networks with decisions, in: First Workshop on fOrmal VERification, Logic, Automata, and sYnthesis, 2019, OVERLAY 2019, Vol. 2509, CEUR-WS.org, 2019, pp. 77–82.
URL http://ceur-ws.org/Vol-2509/paper12.pdf

[38] C. Combi, R. Posenato, L. Viganò, M. Zavatteri, Access controlled temporal networks, in: 9th International Conference on Agents and Artificial Intelligence (ICAART 2017), INSTICC, ScitePress, 2017, pp. 118–131. doi:10.5220/0006185701180131.

[39] C. Combi, R. Posenato, L. Viganò, M. Zavatteri, Conditional simple temporal networks with uncertainty and resources, Journal of Artificial Intelligence Research 64 (2019) 931–985. doi:10.1613/jair.1.11453.

[40] C. Combi, L. Viganò, M. Zavatteri, Security constraints in temporal role-based access-controlled workflows, in: 6th ACM Conference on Data and Application Security and Privacy, CODASPY '16, ACM, 2016, pp. 207–218. doi:10.1145/2857705.2857716.

[41] M. Zavatteri, L. Viganò, Last man standing: Static, decremental and dynamic resiliency via controller synthesis, Journal of Computer Security 27 (3) (2019) 343–373. doi:10.3233/JCS-181244.

[42] M. Zavatteri, L. Viganò, Constraint networks under conditional uncertainty, in: 10th International Conference on Agents and Artificial Intelligence — Volume 2 (ICAART 2018), SciTePress, 2018, pp. 41–52. doi:10.5220/0006553400410052.

[43] M. Zavatteri, L. Viganò, Conditional uncertainty in constraint networks, in: Agents and Artificial Intelligence, Springer, 2019, pp. 130–160. doi:10.1007/978-3-030-05453-3_7.

[44] M. Zavatteri, C. Combi, L. Viganò, Resource controllability of workflows under conditional uncertainty, in: Business Process Management Workshops, Springer, 2019, pp. 68–80. doi:10.1007/978-3-030-37453-2_7.

[45] M. Zavatteri, R. Rizzi, T. Villa, Complexity of weak, strong and dynamic controllability of cncus, in: First Workshop on fOrmal VERification, Logic, Automata, and sYnthesis, 2019, OVERLAY 2019, Vol. 2509, CEUR-WS.org, 2019, pp. 83–88.
URL url={http://ceur-ws.org/Vol-2509/paper13.pdf},

[46] M. Zavatteri, Temporal and resource controllability of workflows under uncertainty, in: Proceedings of the Dissertation Award, Demonstration, and Industrial Track at BPM 2019, Vol. 2420, CEUR-WS.org, 2019, pp. 9–14.
URL http://ceur-ws.org/Vol-2420/paperDA3.pdf

[47] H. A. Kautz, D. A. McAllester, B. Selman, Encoding Plans in Propositional Logic, in: KR '96, Morgan Kaufmann, 1996, pp. 374–384.

[48] H. A. Kautz, B. Selman, Planning as satisfiability, in: ECAI '92, John Wiley & Sons, Inc., 1992, pp. 359–363.

[49] R. Huang, Y. Chen, W. Zhang, A novel transition based encoding scheme for planning as satisfiability, in: AAAI 2010, AAAI Press, 2010.

[50] S. Gocht, T. Balyo, Accelerating SAT based planning with incremental SAT solving, in: ICAPS 2017, AAAI Press, 2017, pp. 135–139.

[51] N. Eén, N. Sörensson, Temporal induction by incremental SAT solving, Electronic Notes in Theoretical Computer Science 89 (4) (2003) 543–560.

[52] C. W. Barrett, R. Sebastiani, S. A. Seshia, C. Tinelli, Satisfiability modulo theories, in: Handbook of Satisfiability, IOS Press, 2009, pp. 825–885.

[53] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, L. E. Kavraki, Incremental task and motion planning: A constraint-based approach, in: Robotics: Science and Systems XII, 2016.
URL http://www.roboticsproceedings.org/rss12/