Information Systems 118 (2023) 102257

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is

Flexible temporal constraint management in modularized processes

Roberto Posenato [*], Carlo Combi

Department of Computer Science, University of Verona, Italy

A R T I C L E   I N F O

Article history:
Received 28 December 2022
Received in revised form 15 July 2023
Accepted 17 July 2023
Available online 22 July 2023

Keywords:
Process-aware information system
Subprocess
Process modularity
Temporal constraint
Temporal flexibility
Controllability

A B S T R A C T

Managing temporal process constraints in modularized processes is an important task, both during the design, as it allows the reuse of temporal (child) process models, and during the checking of temporal properties of processes, as it avoids the necessity of "unfolding" child processes within the main process model. Taking into account the capability of providing modular solutions, modeling and checking temporal features of processes is still an open problem in the context of process-aware information systems.

In this paper, we present and discuss a novel approach to represent flexible temporal constraints in modularized time-aware BPMN process models.

To support temporal flexibility, allowed task durations are represented through *guarded ranges* that allow a limited (guarded) restriction of task durations during process execution if it is necessary to guarantee the satisfaction of all temporal constraints. We, then, propose how to derive a compact representation of the overall temporal behavior of such time-aware BPMN models. Such compact representation of child processes allows us to check the *dynamic controllability* (DC) of a parent time-aware process model without "unfolding" the child process models. Dynamic controllability guarantees that process models can have process instances (i.e., executions) satisfying all the temporal constraints for any possible combination of allowed durations of tasks and child processes. Possible approaches for even more flexibility by solving some kinds of DC violations are then introduced.

We use a real process model from a healthcare domain as a motivating example, and we also present a proof-of-concept prototype confirming the concrete applicability of the solutions we propose, followed by an experimental evaluation.

© 2023 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

1. Introduction

Modularization is an important technique in many software system design and programming fields. It corresponds to the widely acknowledged strategy of decomposing a problem into different subproblems, which are easier to face, and then solving the overall problem by merging the different contributions coming from the solution of the identified subproblems. In the software context, modularization also helps reuse already-found solutions applicable in different contexts. Moving closer to the context of information systems and, in particular, to the design of business processes, such features can also be found in BPMN, where child processes (i.e., subprocesses) can be suitably defined to allow a more straightforward visual understanding of the overall structure of a (parent) business process. Moreover, such child processes (i.e., *reusable subprocesses* or *call activities* [1]) can be reused in different parent processes [2][3].[1]

Thus, the support of complex parent processes, possibly containing different and nested child processes, is a crucial issue for *process-aware information systems (PAIS)* both for the reuse of existing process knowledge from a process repository and for the modular design of business processes.

On the other hand, temporal constraints of processes, such as deadlines, minimum and maximum task duration, and maximum allowable delays between task execution, are becoming of interest in many different research and application contexts [4–12].

In particular, recent research studies have focused on the capability of representing both contingent and requirement constraints. While the first ones are specified through a not modifiable range of possible durations for tasks and are not under the control of the process engine, the second ones have to be managed by the engine, which could, for example, decide about

[*] Corresponding author.
E-mail addresses: roberto.posenato@univr.it (R. Posenato), carlo.combi@univr.it (C. Combi).

[1] We use two distinct pair of squared brackets when there are two citations because we use the notation $[a, b]$ for temporal ranges.

the delay in the enactment of some activities, to satisfy all the other constraints, possibly considering the contingent ones. Recently, to add flexibility to the process design, a new kind of contingent constraint, specified through *guarded link* has been introduced to allow a type of (limited) shrinking of task contingent constraints [9]. The main property considered for such kind of time-aware process models is that of *dynamic controllability* (DC) [4][6]. It guarantees that a process model can be executed according to any possible allowed duration of *all* tasks, specified through guarded links while still satisfying *all* requirement constraints. Dynamic controllability guarantees that the process engine can dynamically manage the satisfaction of the temporal constraints, by "reacting" to the different task durations that are revealed during the process execution.

Apparently, process temporal constraints and process modularity seem disjoint concepts. Indeed they can apparently be tackled in isolation, as the first ones refer to specific constraints that have to be satisfied during the execution of the processes, while the second property refers to the capability of specifying a process model through already specified child processes. However, going further with respect to just scratching the surface, we may discover that modularity, in combination with the reuse of time-aware process models, requires the ability to represent the overall temporal behavior of processes [9][13]. Indeed, the temporal constraints of a process containing time-sensitive child processes can be evaluated in a *truly modular* way only if we can check the overall process without replacing every child process with its (time-aware) components. Moreover, one may then attach temporal properties of reusable (child) process models when storing them in a central time-aware process repository.

To the best of our knowledge, the problem of representing the overall temporal behavior of a reusable and dynamically controllable child process, having both flexible duration ranges and **different possible execution paths**, has not been considered before. Moreover, the task of verifying the overall dynamic controllability of a parent process without leveraging on a suitable representation of the temporal behavior of child processes has not been considered in previous proposals.

- We propose a time extension to (the main constructs of) BPMN, which allows the representation of flexible contingent durations for tasks named *guarded ranges* and different temporal constraints between tasks named *time lags*. Possible different execution paths are represented through suitable exclusive gateways. In this direction, we complete here a previous proposal, where only a restricted fragment of BPMN, mainly considering processes without splitting gateways, was studied [9].
- We propose a new suitable approach to derive a compact representation of the overall temporal behavior, named after *prototypal link with contingency*, of such time-aware BPMN models. To do it, we leverage a recently introduced kind of temporal constraint network [14], by proposing a mapping of BPMN models to *Flexible simple Temporal Networks with Uncertainty* (FTNUs) and then deriving the overall temporal characterization of a BPMN model.
- We discuss how to verify the dynamic controllability of a parent time-aware process model without "unfolding" the child time-aware process models. Indeed, it could be that the overall DC of the parent process can only be guaranteed for a specific "tuning" of the temporal properties of the child processes.
- We discuss different kinds of DC violations and propose some solutions to make the overall process model more flexible, allowing the management of some DC violations to some extent.
- We introduce a proof-of-concept prototype freely available on the Web, implementing the different temporal extensions and related algorithms we propose in this paper, and provide a detailed experimental evaluation of the proposed algorithms.

In detail, the remainder of this work is organized as follows: Section 2 recalls the main concepts of FTNUs. Section 3 introduces a clinical guideline for managing Adult Stroke Emergency as an example of a (parent) process model with child processes and temporal properties. We use this clinical example throughout the paper. Section 4 introduces a time-aware BPMN-based approach where time-aware child processes can be specified. In turn, Section 5 presents all the technical details of the computation of temporal features of the child processes. In particular, it discusses how to characterize time-aware processes by mapping them to corresponding FTNUs. All relevant concepts are formally described, and the introduced theorems are proven. Section 6 then discusses how the temporal properties of a child process can be used to check the controllability of the overall process without unfolding its child processes. Proper management of DC violations is then introduced to deal with further flexibility in managing time-aware processes. Section 7 introduces some tools, freely available on the Web, which support the design of time-aware (child) process models and the verification of their DC-related properties. Section 8 describes the experimental evaluation we did with respect to the proposed algorithms. Section 9 discusses the existing work on managing temporal constraints and modularity in business processes. Finally, Section 10 discusses the main results and the limitations of our work and sketches future research lines. The paper ends with Appendix, which contains the proofs of lemmas we introduced to discuss the DC checking of modularized temporal BPMN models.

2. Background: Flexible simple temporal networks with uncertainty

Flexible simple Temporal Network with Uncertainty (FTNU) is a model of temporal reasoning based on a graph representation of events and temporal constraints [14]. FTNU is an extension of Conditional Simple Temporal Network with Uncertainty (CSTNU) [15][16] where contingent constraints (called *contingent links*) are replaced by guarded constraints (called *guarded links*) [17] to gain greater flexibility.

An FTNU can be represented as a directed weighted graph (see Fig. 2) whose nodes represent the time occurrence of events and whose edges represent temporal constraints between pairs of events.

The nodes are real-valued variables (also called *timepoints*) that have to be assigned. A node is said *executed* when a real value (*execution time*) is set to it. In each FTNU, there is a particular node, Z, that is executed as the first node at time 0. From the point of view of execution, there are two kinds of nodes: *ordinary*, and *contingent*. For ordinary nodes, the execution time is decided by an agent that we call the *execution engine*, while for contingent nodes the value is decided by the *environment* during run time. The environment can also be viewed as an adversary player with respect to the execution engine [18][19].

Some ordinary nodes may also be *observation* timepoints. An observation timepoint is associated with a proposition (boolean variable) in a biunique way. If p is a proposition, we usually denote its observation timepoint by $P?$. A proposition is set to true ($\top$) or false ($\bot$) *by the environment* when its observation timepoint is executed *by the execution engine*. A complete specification of all proposition values of a network is called *scenario*. A scenario is incrementally revealed as observation timepoints are executed.

Given a set $\mathcal{P}$ of propositions, a *(propositional) label* ℓ is any conjunction of literals, where a literal is either a proposition $p \in \mathcal{P}$ or its negation $\neg p$. The *empty label* is denoted by $\boxdot$. The *label universe of* $\mathcal{P}$, denoted by $\mathcal{P}^*$, is the set of all finite labels whose literals are drawn from $\mathcal{P}$. Two labels $\ell_1, \ell_2 \in \mathcal{P}^*$ are *consistent* if and only if their conjunction $\ell_1 \wedge \ell_2$ is satisfiable and a consistent label ℓ_1 *entails* a consistent label ℓ_2 (written $\ell_1 \supset \ell_2$) if and only if all literals in ℓ_2 appear in ℓ_1 too.

Each node has a *propositional label* that determines in which scenarios the node has to be considered during an execution. Suppose that a node label becomes $\perp$ during execution. In that case, it means that the node and all its incident edges (i.e., constraints involving the node) have not to be considered in the current scenario and, therefore, they can be removed from the network. Different executions of the same network may determine different scenarios and, therefore, different sets of nodes to be executed. We represent the label of a node as a subscript label enclosed by square brackets. For example, $A_{[p\neg q]}$ means that node A has to be considered in scenarios where $p\neg q$ is true, i.e., p is $\top$, and q is $\perp$. Node Z has an empty label because it has to be executed in all scenarios. When a label is empty, it can be omitted.

Regarding edges, they represent binary temporal constraints between pairs of nodes that must be satisfied when nodes are executed. There are two kinds of edges. The first one is given by *requirement links*. A requirement link, $X \xrightarrow{[l,u],\,\ell} Y$, represents a lower and an upper bound constraint on the distance between the two timepoints it connects: $l \leq Y - X \leq u$. The interval $[l, u]$ is called the *duration range*. The label ℓ specifies in which scenarios the constraint has to be considered (and satisfied). The other kind is related to contingent timepoints. Each contingent timepoint has exactly one incoming edge of the kind *guarded link*, drawn as a double line. A guarded link $A \xrightarrow{[[x,x'][y',y]]} C$ specifies when the contingent timepoint C can occur with respect to A, an ordinary timepoint also called *activation* timepoint; the range of a guarded link consists of a duration range $[x, y]$ augmented with two *guards*, the *lower guard* x' and the *upper guard* y' that determine the *core* of the link [17]. Before executing the activation timepoint, the duration range $[x, y]$ can be modified by the execution engine. However, any modification has to be accomplished in a way respecting the corresponding guards, i.e., $x \leq x'$ and $y \geq y'$. When the activation timepoint is executed, the current value $[x^*, y^*]$ of the duration range becomes *contingent range*, i.e., a range made available to the environment for executing timepoint C and, therefore, not more modifiable. This means that once A is executed, C is guaranteed to be executed by the environment such that $C - A \in [x^*, y^*]$ is held. The specific time at which C is executed is *uncontrollable* since the environment decides it (it can only be observed when it happens). For each guarded link, it is reasonable to assume that the contingent timepoint occurs in the same scenarios as the activation timepoint; therefore, we assume that both nodes have the same label. The label of the guarded link is omitted because it is implicit by the label of its endpoints and because between the two endpoints, no other constraints are possible.

Definition 1 (*[14]*)**.** A *Flexible simple Temporal Network with Uncertainty* (FTNU) is a tuple $(\mathcal{T}, \mathcal{P}, \mathcal{OT}, O, L, \mathcal{C}, \mathcal{G})$, where

- $\mathcal{T}$ is a set of real-valued variables. These variables are the *timepoints* of the network. $\mathcal{T}$ always contains the timepoint Z that is assumed to be the first timepoint to be executed, i.e., its value is set by the executing agent;
- $\mathcal{P} = \{p, q, r, s, \dots\}$ is a finite set of propositional letters.
- $\mathcal{OT} \subseteq \mathcal{T}$ is a set of *observation timepoints*.
- $O : \mathcal{P} \rightarrow \mathcal{OT}$ is a bijection that, given a propositional letter, assigns a unique observation timepoint to it. The truth value of a proposition is set by the environment when its observation timepoint is executed. Usually, if q is a proposition, its observation timepoint has the name $Q?$.
- $L : \mathcal{T} \rightarrow \mathcal{P}^*$ is a function that assigns a propositional label to each timepoint $X \in \mathcal{T}$. A (propositional) label is a conjunction of propositional letters. A true-valued label of a node indicates that the node has to be executed.
- $\mathcal{C}$ is a set of *labeled requirement links*. Each requirement link is denoted as $(u \leq Y - X \leq v, \alpha)$ or as $X \xrightarrow{[u,v],\,\alpha} Y$; $X, Y \in \mathcal{T}$, $u, v \in \mathbb{R}$ with $u \leq v$, $0 \leq v$, and $\alpha \in \mathcal{P}^*$. A requirement link has to be satisfied when its label has (will have) a true value. In other words, a requirement link can be ignored only when its label becomes false during execution.
- $\mathcal{G}$ is a set of *guarded links*. Each guarded link is represented as $(A, [[x, x'][y', y]], C)$ or as $A \xrightarrow{[[x,x'][y',y]]} C$; A and C are timepoints, called *activation* and *contingent* timepoints, respectively; $x, y \in \mathbb{R}$ are the external bounds; $x', y' \in \mathbb{R}$ are the *guards*. $[x, y]$ is called *external range* of the guarded link. It must hold $0 < x \leq x' \leq y' \leq y < \infty$, and $L(A) = L(C)$. Moreover, if $(A_i, [[x_i, x'_i][y'_i, y_i]], C_i)$ and $(A_j, [[x_j, x'_j][y'_j, y_j]], C_j)$ are two different guarded links in $\mathcal{G}$, C_i and C_j will be distinct timepoints.
- For each labeled constraint $(u \leq Y - X \leq v, \alpha)$, $\alpha \supset L(Y) \wedge L(X)$. This property is called *constraint label coherence* [20].
- For each literal q or $\neg q$ appearing in α, $\alpha \supset L(O(q))$. Such a property is called *constraint label honesty* [20].
- For each $Y \in \mathcal{T}$, if literal q or $\neg q$ appears in $L(Y)$, then $L(Y) \supset L(O(q))$, and $O(q)$ has to occur before Y, i.e., $(\epsilon \leq Y - O(q) \leq +\infty, L(Y)) \in \mathcal{C}$ for some $\epsilon > 0$. Such a property is called *timepoint label honesty* [20].

An FTNU is *dynamically controllable (DC)* if there exists a strategy for executing the timepoints in the network such that all constraints are guaranteed to be satisfied no matter how the durations of the guarded links turn out and no matter how the observations of the various propositions turn out, in real-time—paying attention to the fact that in any given scenario, only the timepoints whose labels are true in that scenario need to be executed, and only the constraints whose labels are meaningful in that scenario need to be satisfied.

The DC-checking decision problem (DC-checking problem) determines whether the given FTNU instance admits a dynamic execution strategy. In [14], the authors showed that the DC-checking decision problem (DC-checking problem) of an FTNU is PSPACE-complete and proposed a DC-checking algorithm based on a constraint propagation approach that:

- determines the minimal constraints between Z and any other timepoints when the instance is DC; such minimal constraints allow the determination of the minimal range of allowed execution times for each timepoint of a DC instance.
- can be used by any scheduler to execute a DC instance, exploiting the dynamic controllability.

Moreover, they provided a proof-of-concept implementation of the algorithm, showing that it is practical for small-size networks (up to 150 nodes and 6 observations).

To check the DC property, an FTNU graph is transformed into an equivalent *distance graph* $\mathcal{D} = (\mathcal{T}, \mathcal{E})$ where the nodes are the same as the original instance. In contrast, the edges are not labeled by labeled ranges, but by labeled values, as detailed below.

Distance graph nodes. Nodes in the distance graph have no propositional labels. Such label removal does not imply loss of information, and any result determined in a distance graph is valid also in the original graph [14].

Distance graph edges. Each link in the FTNU graph is transformed into two or four different edges in the distance graph. In particular, each requirement link $A\xrightarrow{[x,y],\,\ell}B$ is represented as two *ordinary edges* in $\mathcal{E}$:

- $A\xrightarrow{\langle y,\,\ell\rangle}B$, representing constraint $(B-A\le y,\ell)$; and
- $A\xleftarrow{\langle -x,\,\ell\rangle}B$ for representing constraint $(B-A\ge x,\ell)=(A-B\le -x,\ell)$,

where $x,y\in\mathbb{R}$ and $\ell\in\mathcal{P}$.

Each guarded link $A\xrightarrow{[[x,x'][y',y]]}C$ is represented in $\mathcal{E}$ as:

- two *ordinary edges* for representing the duration range $[x,y]$: $A\xrightarrow{\langle y,\,\ell\rangle}C$ and $A\xleftarrow{\langle -x,\,\ell\rangle}C$ where ℓ is the label of node A;
- two new other labeled edges, called *lower* and *upper-case edges* for representing the two guards x' and y':
  - A lower-case edge, $A\xrightarrow{\langle c:x'\rangle}C$, expresses that C cannot be forced to be executed at a time greater than x' after A, i.e., it is not possible to add a constraint $A\xleftarrow{\langle -x^*,\,\ell\rangle}C$, $x'<x^*$ to the network.
  - An upper-case edge, $A\xleftarrow{\langle C:-y'\rangle}C$, expresses that C cannot be forced to be executed at a time less than y' after A, i.e., it is not possible to add a constraint $A\xrightarrow{\langle y^*,\,\ell\rangle}C$, $y^*<y'$ to the network.

The FTNU DC-checking algorithm introduces and modifies ordinary edges by suitable propagation rules and can use two new kinds of edges, named after *conjunct-lower-case edges* and *conjunct-upper-case edges*. Both the conjunct-lower-case edges and the conjunct-upper-case ones can be viewed as special upper/lower bounds to execute timepoints.

In particular, a conjunct-lower-case edge is represented as $Z\xrightarrow{\langle\daleth:v,\,\ell\rangle}X$, where Z is the origin, X is a generic timepoint, $\daleth$ is a list of contingent timepoint names, v is a nonnegative value, and ℓ is a propositional label. Intuitively, given a timepoint X, the conjunct-lower-case edge $Z\xrightarrow{\langle\daleth:v,\,\ell\rangle}X$ represents the *lowest upper bound* for X that can be set between Z and X in scenarios where ℓ is true and must be satisfied when all the contingent timepoints referenced in $\daleth$ are executed at their lower guarded time. Suppose a requirement constraint $Z\xleftarrow{\langle -w,\,\ell\rangle}X$ with $w>v$ is added to the FTNU. In that case, the network cannot be DC because this new constraint is not satisfied, at least in a possible execution of the network, i.e., in an execution where the contingent timepoints in $\daleth$ occur at their lower guarded time in one of the scenarios where ℓ is true. Such an execution is possible because all contingent timepoints can always occur at their lower guarded time.

Analogously, a conjunct-upper-case edge is represented as $Z\xleftarrow{\langle\aleph:-v,\,\ell\rangle}X$, where Z is the origin, X is a generic timepoint, $\aleph$ is a list of contingent timepoint names, v is a nonnegative value, and ℓ is a propositional label. Given an X, the conjunct-upper-case edge $Z\xleftarrow{\langle\aleph:-v,\,\ell\rangle}X$ represents the *greatest lower bound* of A that can be set between Z and X, in scenarios where ℓ is true and has to be satisfied when all the involved contingent timepoints are executed at their *upper guarded time*. If an ordinary constraint $Z\xrightarrow{\langle w,\,\ell\rangle}X$ with $w<v$ is added to the FTNU, then the network cannot be DC.

A detailed description of the behavior and soundness of the rules, as well as the completeness of the FTNU DC-checking algorithm, can be found in [14]. Its time complexity is $O(M|\mathcal{T}|^4 3^{|\mathcal{P}|}2^{|\mathcal{G}|})$, where M is the maximum absolute value of any weight in the network [14,16,21].

3. Motivating example

We consider a high-level specification of an excerpt from a clinical guideline related to the management of Adult Stroke Emergency [22] as a motivating scenario. A possible BPMN model of this process is depicted in Fig. 1(a) (adapted from [22, page S819]).

Example 1. The parent process starts with four sequential child processes: *Stroke Recognition* (child process P_1: SR), *Prehospital Management* (child process P_2: PM), *General and Neurologic Assessment* (child process $P_{3\&4}$: G&NA), *Imaging Computed-Tomography Scan* (child process P_5: ICTS). Such child processes correspond to the first five boxes discussed in [22] to acquire all the information needed for the following decision-making actions.

After child process P_5, two alternative paths are possible according to whether *Imaging Computed-Tomography Scan* shows a hemorrhage. If there is a haemorrhage, (*yes* branch), child process *Consult Neurologist or Neurosurgeon* (P_7: CN) is performed, followed by child process *Haemorrhage Management* (P_{11a}: HM) where the haemorrhage patient is cared for. Otherwise, (branch labeled by *no*), child process *Consider Fibrinolysis* (P_6: CF) is performed to determine if the patient is eligible for fibrinolytic therapy. Afterward, two different paths are possible according to the evaluation performed through P_6. If the patient is suitable for fibrinolysis, drug *Recombinant Tissue Plasminogen Activator (rtPA)* is administered to the patient (activity T_{10}: rtPA) and, then, *Post rtPA Management* (P_{12}: PrtPAM) must be performed. If the patient cannot afford fibrinolysis, a therapy based on Aspirin is administered (child process P_9: Asp) and, then, a *Stroke Management* child process is executed (P_{11b}: SM).

Finally, child process *Stroke Unit Admission* (P_{13}: SUA) is executed for any possible previous path.

Among all the child processes present in Fig. 1(a), we consider the schema of child processes P_2 and P_6 as depicted in Figs. 1(b) and 1(c), respectively.

P_2 starts with an *initial assessment, considering also symptom onset* (T_0: Initial & Sym Onset Asmt) of the patient. If the patient is hypoxemic, then the oxygen supply has to be set (task T_1: SetOxigenSupply). Then, (possibly) four different tasks have to be executed in any order with possible overlaps, namely setting the neurological monitoring (task T_2: SetNeuroMonitor), checking the glucose (task T_3: BloodGlucCheck, only if the patient is hypoglycemic), doing a prehospital notification (T_4: PreHospNotify) and performing a transfer to hospital (task T_5: TransferToHosp).

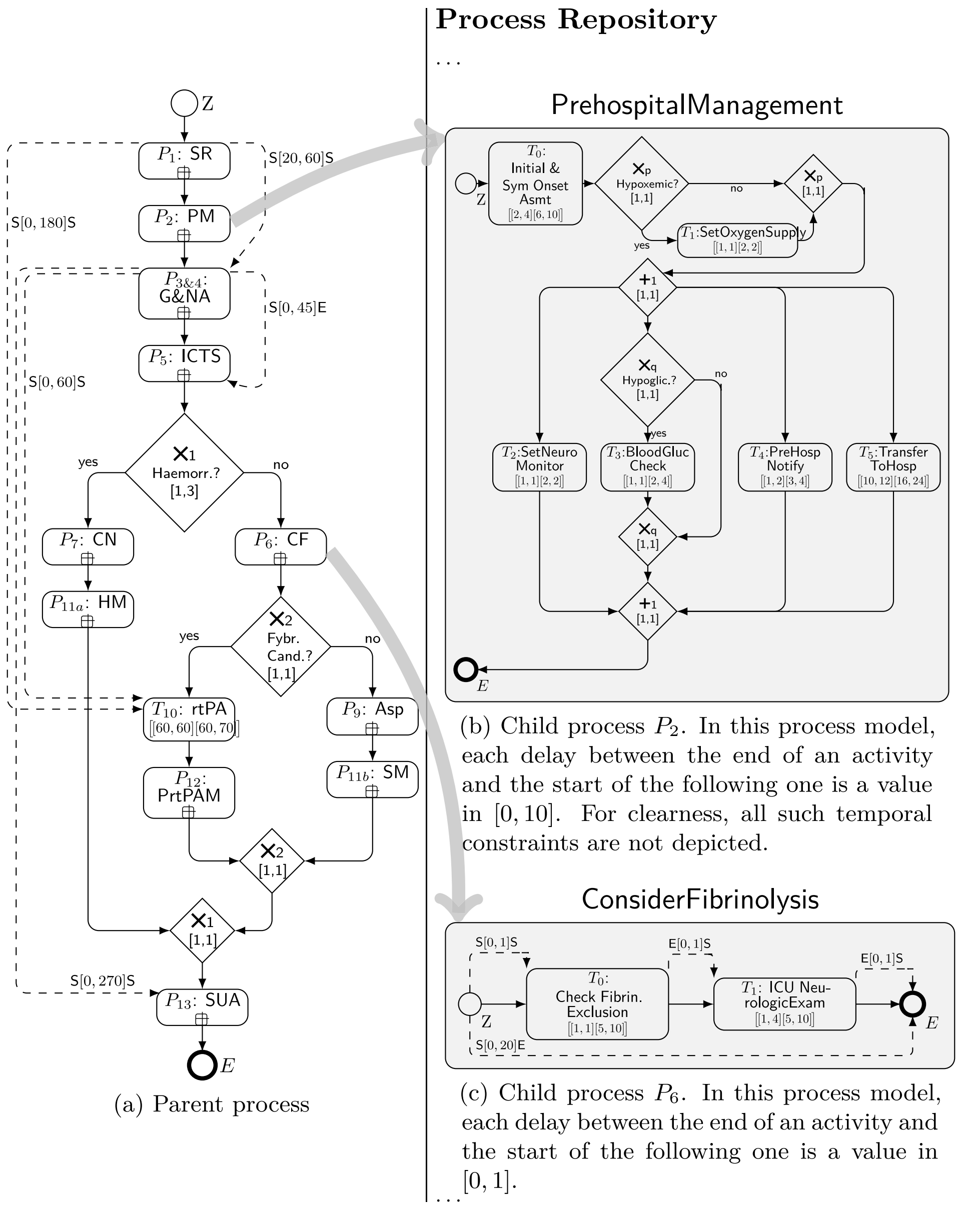
P_6 consists of checking fibrinolytic exclusion followed by a fast neurologic exam, represented by the sequential execution of tasks T_0: Check Fibrin. Exclusion and T_1: ICU NeurologicExam, respectively.

All the presented schemata have been enriched with temporal constraints – following the concepts introduced in [9] – that must be satisfied to guarantee the clinically successful completion of each process step. The values of such temporal constraints were set according to the guideline specification and the temporal constraints – graphically represented in the diagram in [22, page S819] – while for the prehospital management, we considered the temporal ranges discussed in [23].

Such temporal constraints will help clinical stakeholders plan their work, as they know how long previous steps will take and how much freedom they have to perform their clinically relevant tasks.

The time spans of tasks and child processes are not entirely under the control of the PAIS as clinicians carry out these tasks. Therefore, task durations are represented as *guarded ranges*. Such duration ranges may be *partially* restricted by the process engine during execution to ensure the successful completion of the processes.

Example 2. Task T_0 of child process P_2 is associated to a guarded range, represented as $[[2,4][6,10]]$. It means that T_0 will have a


Process Repository
Z
P_1: SR
S[20, 60]S
S[0, 180]S
P_2: PM
$P_{3\&4}$: G&NA
S[0, 45]E
P_5: ICTS
S[0, 60]S
$\times_1$ Haemorr.? [1,3]
yes
no
P_7: CN
P_6: CF
P_{11a}: HM
$\times_2$ Fybr. Cand.? [1,1]
T_{10}: rtPA [[60, 60][60, 70]]
P_9: Asp
P_{12}: PrtPAM
P_{11b}: SM
$\times_2$ [1,1]
$\times_1$ [1,1]
S[0, 270]S
P_{13}: SUA
E
PrehospitalManagement
T_0: Initial & Sym Onset Asmt [[2, 4][6, 10]]
$\times_p$ Hypoxemic? [1,1]
$\times_p$ [1,1]
T_1:SetOxygenSupply [[1, 1][2, 2]]
$+_1$ [1,1]
$\times_q$ Hypoglic.? [1,1]
T_2:SetNeuro Monitor [[1, 1][2, 2]]
T_3:BloodGluc Check [[1, 1][2, 4]]
T_4:PreHosp Notify [[1, 2][3, 4]]
T_5:Transfer ToHosp [[10, 12][16, 24]]
$\times_q$ [1,1]
$+_1$ [1,1]
ConsiderFibrinolysis
S[0, 1]S
E[0, 1]S
E[0, 1]S
T_0: Check Fibrin. Exclusion [[1, 1][5, 10]]
T_1: ICU NeurologicExam [[1, 4][5, 10]]
S[0, 20]E

(a) Parent process

(b) Child process P_2. In this process model, each delay between the end of an activity and the start of the following one is a value in $[0, 10]$. For clearness, all such temporal constraints are not depicted.

(c) Child process P_6. In this process model, each delay between the end of an activity and the start of the following one is a value in $[0, 1]$.

Fig. 1. Motivating example: The process model for managing Adult Stroke Emergency.

duration (initially set) in a range between 2 and 10 time units. However, before the execution of the task, such allowed duration range can be restricted. Still, in any case, the lower bound of the range must not exceed 4 min while the upper bound cannot be set below 6 (e.g., a duration range of $[3, 5]$ or $[5, 8]$ is disallowed). Physicians in charge of T_0 may take between 2 and 10 min to perform it normally. However, the range can be shrunk to $[4, 6]$ at maximum before enacting T_0 if the process engine determines that it is necessary to guarantee the satisfaction of all other temporal constraints. In any case, the physician will know the allowed range before starting the task and, once the task is started, the range will not be modified anymore. This ensures that the user who executes the task has sufficient flexibility to complete the task successfully.

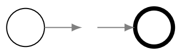
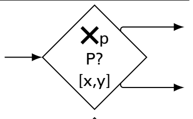
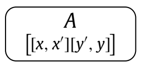
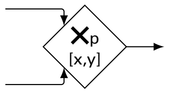
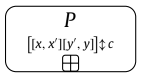
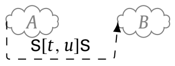
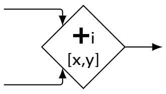
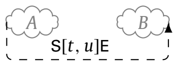
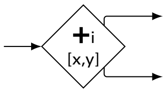
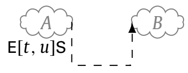
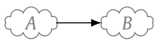
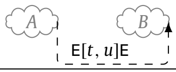
Constraints on gateways constitute standard temporal constraints, specifying the possible durations (within a range) that are under the PAIS's control. It means that the PAIS has some time allowed to manage its internal task, such as moving on the control (and the log) of the execution, managing the resources for the following tasks, evaluating the chosen execution path, and so on. Gateway $\times_1$ Haemorr.? has a temporal constraint $[1, 3]$ on its allowed duration. Before the gateway execution, if the process engine needs to restrict its duration range to guarantee the successful execution of the process, it can restrict it even to a single value like $[1, 1]$ or $[3, 3]$.

Moreover, further temporal constraints, the PAIS has to consider, can be specified between start/end timepoints of different tasks (through labeled dashed edges). The time distance (i.e., duration) between the start of process P_1 and the start of process $TP_{3\&4}$ has to be between 20 and 60 min (expressed as S[20, 60]S).

As already discussed, two challenges emerge in this context.

- The first challenge concerns the representation of the overall temporal behavior of (child-)processes. One must be able to describe how a child process, for example, Prehospital Management, behaves temporally if it is reusable within any time-aware processes. (Prehospital Management is common to other ICU processes.) In [9], this issue was considered and solved only for subprocesses without alternative paths.

Table 1
Time-aware BPMN constructs. Clouds stand as meta-symbols for any component of the process schema, i.e., task, gateway or child process.

(a)	Start/End event		(g)	ORsplit	
(b)	Task		(h)	ORjoin	
(c)	Child process (subprocess)		(i)	start-start	
(d)	ANDjoin		(j)	start-end	
(e)	ANDsplit		(k)	end-start	
(f)	Sequence-Flow Edge		(l)	end-end	

However, the presence of alternative paths poses some challenges to the compact representation of allowed temporal ranges for the duration of the subprocess.

- The second challenge is related to the efficient temporal analysis of the parent process. Ideally, this analysis should be accomplished without unfolding all considered child processes to make it effective and quick. In general, the challenge is to find the possible temporal configurations of all child processes that allow the successful execution of the parent process. Although this problem received some attention in [9] only for single-path child processes and only single-path parent processes, no algorithms have been proposed in the literature dealing with the above challenge. Here, we consider more general models where multiple alternative paths may be present in the parent and child processes, and we propose an algorithm that discovers all possible temporal configurations of the child process to use in the parent process. For example, assuming to have a temporal representation of all child processes in Fig. 1, the proposed algorithm discovers that a possible configuration of P_2 and P_6 with guarded ranges $[[16, 28][28, 28]]$ and $[[2, 8][11, 11]]$, respectively, guarantees a successful execution of the parent process.

4. Modeling temporal features of BPMN modularized processes

In this section, we propose a temporal extension of BPMN that allows the representation of simple modularized time-aware BPMN processes. Then, we propose the notion of dynamic controllability for time-aware BPMN processes.

4.1. Extending BPMN constructs with temporal properties and constraints

BPMN processes can represent different temporal aspects, adding a temporal dimension to a relevant subset of BPMN elements and suitable temporal constraints based on concepts presented in [4,8,24]. The obtained *time-aware* BPMN encourages the temporal characterization of tasks, gateways, and time lags between process elements, as depicted in Table 1. The constructs of such time-aware BPMN extend those introduced in [9], where conditional branches, represented through OR split/join, were not considered.

While Table 1 depicts the constructs of time-aware BPMN, we describe the introduced temporal aspects by considering the example of Fig. 1 and borrowing the notions of "activity" and "gateway" from the BPMN standard [1].

– `Activities` can be distinguished into *tasks*, which are atomic (not decomposable), and *child processes (or subprocesses)*, which can be decomposed into smaller parts, like child processes, tasks, gateways, etc.

Tasks (see Table 1.b) have a duration attribute represented as a range $[[x, x'][y', y]]$, with $0 < x \leq x' \leq y' \leq y < \infty$, where x/y is the minimum/maximum allowed time span for an activity to go from state "started" to "completed" [25]. Here, we assume that all given values are represented in a predefined time unit (e.g., minutes in our example). At run-time, the process engine can modify the values of x and/or y to ensure successful execution, but observing the constraint that $x \leq x'$ and $y' \leq y$, where x' and y' represent the lower/upper guard, respectively, and are fixed by the process designer. Moreover, the process engine cannot fix the real duration of an activity but only observe after who is in charge of executing it completes the activity (*contingent duration*). The process engine considers the real duration to enact the following elements properly. Who is in charge of executing the activity must observe the two bounds x and y at the enacting phase.

The temporal features of child processes are derived from those of their composing activities, considering all the time constraints specified within the processes themselves. As we will see in detail in Section 5.3, guarded ranges are not enough to specify the temporal behavior of a child process, and we also need to set the *contingency* of the subprocess (see Table 1.c) [9].

– `Sequence-Flow Edges` enable the execution of the second component B, when the previous one, A ends. A and B can represent activities and gateways. While the end of A may be out of the control of the process engine, when, for example, A is a task or a child process, the start of B is always under the control of the process engine. The delay by which B starts is determined by the process engine considering all the current temporal constraints. A designer can limit such a delay by setting a $E[x, y]S$ *time lag* (see below).

– `Gateways` also have a duration range of the form $[l, u]$, with $0 \leq l \leq u \leq \infty$. The process engine plans the actual execution time for such elements by choosing a suitable value of the range, according also to the requirements of its internal tasks. If a designer does not set a duration, it is assumed to be $[0, \infty]$.

– `Time Lags` are depicted in Fig. 1 as dashed edges that connect any two components of the process [4]. Time lags limit the time distance between the starting/ending instants of two components and have the form $I_S[u, v]I_F$, where I_S is the starting (S)/ending (E) instant of the first component, while I_F is the starting/ending instant of the second one [4].

From now on, we consider only well-structured processes (i.e., processes where split and join gateways form single coherent single-entry single-exit (SESE) blocks) as they offer several advantages in terms of comprehension, modularity, and robustness [8,26, for a detailed discussion]. Moreover, as for the represented temporal constraints, we assume that time lags are defined between components when they belong to the same execution path.

4.2. Controllability of a time-aware BPMN process

Considering temporal aspects, *executing* a time-aware BPMN process means:

1. to schedule the starting time of all elements,
2. to set the duration of gateways, and
3. to determine which are the (alternative) paths to follow during the execution, according to the conditions checked in XOR split gateways.

The duration of a gateway within the given time range is set by the engine and represents the duration the process engine uses to suitably coordinate and move forwards the execution. The values of split conditions (see $P?$ in Table 1) are not known in advance as they are incrementally revealed over time as corresponding tasks are executed. Similarly, the durations of activities are only known as the activities complete. Therefore, a *dynamic* execution of the process must *react* to conditions and contingent durations in real time.

The *reaction* consists in

1. adjusting all the duration ranges of future activities considering the execution times of already finished activities,
2. evaluating which components to execute (or to enact in case of tasks or child processes), and
3. choosing an execution time for the gateways that must be executed as decided in the previous item.

For enacted tasks/child processes, the process engine can only observe the execution time after that the tasks/child processes are finished.

A *viable* execution guarantees that all *relevant* constraints – those holding in the paths being executed – will be satisfied no matter which condition outcomes and durations are revealed over time. A time-aware BPMN process with a dynamic and viable strategy is called *dynamically controllable* (DC).

Checking the dynamic controllability of processes will be the main focus of this paper. More precisely, we first need to be able to

- represent the overall temporal behavior of a (child) process;
- check the overall controllability of processes possibly containing child processes.

5. A computational approach to time-aware processes containing alternative execution paths

The scope of this section is to show how to computationally derive the temporal features of a (child-)process. For this purpose, we consider a process model P with a single start and a single end node.

In Section 5.1, we show how to represent a process model P as an FTNU instance. Sections 5.2 to 5.4 introduce the concepts of prototypal links, contingency, and prototypal links with contingency, to compactly represent the temporal properties of a process model.

5.1. FTNU representation of a process model

The dynamic controllability of a process model P can be verified by converting P into an equivalent FTNU instance S_P using the transformation rules depicted in Tables 2 and 3, and, then, checking the DC property of S using the DC checking algorithm for FTNUs [14].

Tables 2 and 3 show the mapping between the time-aware elements we considered (i.e., tasks, child processes, sequence flow edges, AND gateways, OR gateways, and temporal constraints) and "equivalent" FTNU fragments.

Before showing each element mapping, let us specify how a propositional label for an element in S_P is built considering OR gateways and paths in P.

5.1.1. Label determination

Since a time-aware process may contain OR gateways, different execution scenarios are possible according to which OR branches are executed. Combining all OR gateways execution values, i.e., the truth value of the corresponding proposition determines an execution *scenario*.

A propositional label ℓ may specify only some literals. In such a case, the label represents all scenarios where such literals are $\top$. For example, in a process model having three OR gateways that determine the values of propositions p, q, and r, the label $\ell = p\neg q$ represents scenarios $p\neg qr$ and $p\neg q\neg r$.

To determine the right propositional label ℓ for an element A' in S_P corresponding to an element A in P, it is sufficient to consider the (possible) OR gateways that are present in the path, Π, from the Start event to A in P:

(i) At Start Event, set $\ell = \boxdot$;
(ii) For each (possible) OR Split $\times_i$ in Π associated with proposition p, add p or $\neg p$ to ℓ according to the branch present in Π;
(iii) For each (possible) OR Join $\times_i$ in Π associated with proposition p, remove p literal from ℓ.

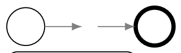
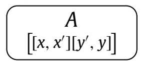
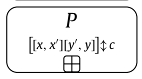
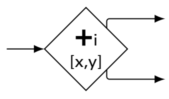
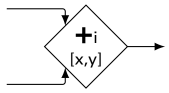
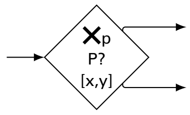
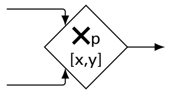
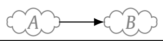
Since we considered only well-structured process schemata, it is not possible that A should be considered in scenarios not represented by the propositional label ℓ. Indeed, a scenario not represented by ℓ, ℓ', has to contain at least one literal that is negated in ℓ. Let us suppose that ℓ' contains q, while ℓ contains $\neg q$. If $\neg q$ is in ℓ, by construction, A is present in the $\perp$ branch of OR Split associated with q. Therefore, it is not possible that A has to be also executed in the $\top$ branch of the same OR Split as dictated by ℓ'.

5.1.2. BPMN mapping to FTNU

Theorem 1. *Given a time-aware process schema P, there exists one and only one FTNU S_P derived by the mapping rules depicted in Tables 2 and 3. S_P can be determined in quadratic time with respect to the number of time-aware-process elements. Moreover, P is dynamically controllable if and only if S_P is dynamically controllable.*

Proof. Now, let us consider the mapping from each time-aware BPMN element to the corresponding FTNU fragments.
– `Start/End event`. These elements are transformed into two FTNU timepoints, Z and E, respectively, connected to the rest of the nodes by an outgoing/incoming requirement link. Such requirement links must have an empty propositional label.
– `Task`. Each task A is converted into two FTNU timepoints, A_S and A_E, representing its start- and end-instants. The duration attribute of A, $[[x, x'][y', y]]$, is converted to the guarded link $(A_S, \big[[x, x'][y', y]\big], A_E)$. A guarded link does not have a propositional label by definition.
– `Child process`. The conversion is analogous to the one of `Task`. The main difference is that the temporal representation of a child process is made by an extended contingent duration, which we introduce in Sections 5.2 to 5.4. Given such an extended duration, by a simple rule, it is possible to determine a (standard) task duration attribute and, therefore, to transform it as done for a task. The rule is summarized in Table 2 and explained in Section 5.4.

Table 2
FTNU transformation rules for business process constructs.

Process	Model	FTNU fragments
Start/End event		$Z \xrightarrow{[0,\infty],\boxdot}$ $\xrightarrow{[0,\infty],\boxdot} E$
Task		$A_S \xRightarrow{[[\mathbf{x},\mathbf{x}'][\mathbf{y}',\mathbf{y}]]} A_E$
Childprocess		$P_S \xRightarrow{[[\mathbf{x},\mathbf{x}^*][\mathbf{y}^*,\mathbf{y}]]} P_E$ where $x \le x^* \le x'$, $y' \le y^* \le y$, and $y^* - x^* \ge c$
ANDsplit		$\begin{tikzcd} & & & {} \\ {} \arrow[r, "{[0,\infty],\ell}"] & +_{i_S} \arrow[r, "{[x,y],\ell}"] & +_{i_E} \arrow[ur, "{[0,\infty],\ell}"] \arrow[dr, "{[0,\infty],\ell}"'] & \\ & & & {} \end{tikzcd}$
ANDjoin		$\begin{tikzcd} {} \arrow[dr, "{[0,\infty],\ell}"] & & & \\ & +_{i_{Sj}} \arrow[r, "{[x,y],\ell}"] & +_{i_{Ej}} \arrow[r, "{[0,\infty],\ell}"] & {} \\ {} \arrow[ur, "{[0,\infty],\ell}"'] & & & \end{tikzcd}$
ORsplit		$\begin{tikzcd} & & & {} \\ {} \arrow[r, "{[0,\infty],\ell}"] & \times_{S+p} \arrow[r, "{[x,y],\ell}"] & \times_{E+p} \arrow[ur, "{[0,\infty],\neg p\ell}"] \arrow[dr, "{[0,\infty],p\ell}"'] & \\ & & & {} \end{tikzcd}$
ORjoin		$\begin{tikzcd} {} \arrow[dr, "{[0,\infty],p\ell}"] & & & \\ & \times_{S-p} \arrow[r, "{[x,y],\ell}"] & \times_{E-p} \arrow[r, "{[0,\infty],\ell}"] & {} \\ {} \arrow[ur, "{[0,\infty],\neg p\ell}"'] & & & \end{tikzcd}$
Sequence-Flow Edge		$\begin{tikzcd} A_S \arrow[r, squiggly] & A_E \arrow[r, "{[0,\infty],\ell}"] & B_S \arrow[r, squiggly] & B_E \end{tikzcd}$

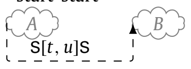
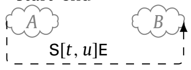
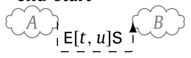
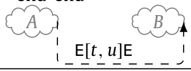
Elements in gray represent complementary information corresponding to elements that must be fixed when fragments are used. ℓ is the scenario in which the fragment is considered.

– `ANDsplit/ANDjoin` gateways. The conversion process is analogous to the one of a task. However, the duration $[x, y]$ is converted to a requirement link $+_{i_S} \xrightarrow{[x,y],\ell} +_{i_E}$ for the i-AND split, and $+_{i_{Sj}} \xrightarrow{[x,y],\ell} +_{i_{Ej}}$ for the i-AND join as the process engine executes control connectors.
– `ORSplit/ORJoin` gateways. The conversion is analogous to the one of an `ANDSplit/ANDJoin` as regards its duration attribute. As for propositional labels, as discussed already, `ORSplit/ORJoin` are the only constructs where labels are modified.
– `Sequence-Flow Edge`. A sequence-flow edge connects two elements of a time-aware process. It states a precedence relation in the execution of the connected elements. It is always converted to a requirement link having $[0, \infty]$ duration. The label ℓ represents the scenario where the two elements are present and is determined according to the approach discussed previously.
– `Time Lags` (see Table 3). Consider a time lag $\langle \mathsf{I_F} \rangle [\mathsf{t}, \mathsf{u}] \langle \mathsf{I_S} \rangle$ between tasks A and B, where $\mathsf{I_F}$ and $\mathsf{I_S}$ represent the kind of instants to be considered, i.e., `S` for the start instant and `E` for the end instant, respectively. Such time lag is converted to a requirement link between the timepoints associated with instants A_{I_F} and B_{I_S} of A and B, respectively. According to the previous label determination, the resulting requirement link has the same duration range $[t, u]$ and label ℓ.

Considering the definitions of time-aware process dynamic controllability (see Section 4.2) and of FTNU dynamic controllability (see Section 2), the dynamic controllability of a process model P implies dynamic controllability of the corresponding FTNU S_P, and vice versa.

Furthermore, the mapping of a process model P into the corresponding FTNU S_P may be calculated considering each element (Activity/Gateway/Sequence Flow/Time Lag) in P making a depth-first visit from the Start Event element. In this way, the mapping can be performed in quadratic time order with respect to the number of P components. Indeed, a process model P corresponds to a graph where nodes are either activities or gateways, while

Table 3
FTNU transformation rules for time lags. Elements in gray represent complementary information corresponding to elements that must be fixed when fragments are used.

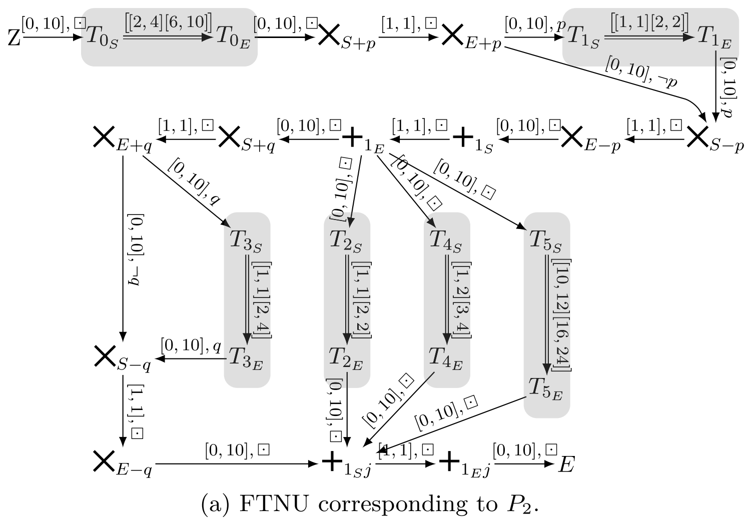
Process model	FTNU fragments
Time lag	
start-start 	$\begin{tikzcd} A_S \arrow[r, squiggly] \arrow[rr, bend right, "{[t,u],\ell}"'] & A_E & B_S \arrow[r, squiggly] & B_E \end{tikzcd}$
start-end 	$\begin{tikzcd} A_S \arrow[r, squiggly] \arrow[rrr, bend right, "{[t,u],\ell}"'] & A_E & B_S \arrow[r, squiggly] & B_E \end{tikzcd}$
end-start 	$\begin{tikzcd} A_S \arrow[r, squiggly] & A_E \arrow[r, "{[t,u],\ell}"'] & B_S \arrow[r, squiggly] & B_E \end{tikzcd}$
end-end 	$\begin{tikzcd} A_S \arrow[r, squiggly] & A_E \arrow[rr, bend right, "{[t,u],\ell}"'] & B_S \arrow[r, squiggly] & B_E \end{tikzcd}$

sequence flows and time lags are directed edges. Thus, a depth-first visit can be performed in $O(|N| + |E|) = O(|N|^2)$, where N is the set of nodes and E is the set of edges. □

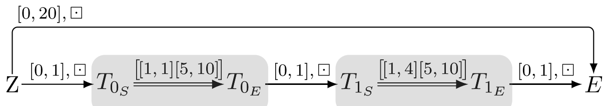
Example 3. Fig. 2 shows the conversions of the child process P_2 and P_6 presented in Fig. 1. Regarding the child process P_2, the proposition associated with the Hypoxemic check is p, while the proposition associated with the Hypoglycemic check is q.

5.2. Prototypal links as an extension of guarded links

Suppose an FTNU S_P associated with the time-aware BPMN process P is DC.



(a) FTNU corresponding to P_2.



(b) FTNU corresponding to P_6.

Fig. 2. FTNUs corresponding to subprocesses P_2 and P_6 in Fig. 1. The shadow regions are added only to emphasize nodes relative to tasks.

Table 4
Constraints between Z and E of S_{P_2}, derived by the DC-checking algorithm.

Lower bounds	$Z \xleftarrow{\langle -16, \neg p\rangle} E$ $Z \xleftarrow{\langle -17, p\rangle} E$
Conjunct lower-case bounds	$Z \xrightarrow{\langle t_0t_2:79, \neg p\rangle} E$, $Z \xrightarrow{\langle t_0:80, \neg p\rangle} E$, $Z \xrightarrow{\langle t_2:85, \neg p\rangle} E$, $Z \xrightarrow{\langle t_0t_1t_2:90, p\rangle} E$, $Z \xrightarrow{\langle t_0t_2:91, p\rangle} E$, $Z \xrightarrow{\langle t_0t_1:91, p\rangle} E$, $Z \xrightarrow{\langle t_0:92, p\rangle} E$, $Z \xrightarrow{\langle t_1t_2:96, p\rangle} E$, $Z \xrightarrow{\langle t_2:97, p\rangle} E$, $Z \xrightarrow{\langle t_1:97, p\rangle} E$
Conjunct upper-case bounds	$Z \xleftarrow{\langle T_1:-18, p\rangle} E$, $Z \xleftarrow{\langle T_0:-20, \neg p\rangle} E$, $Z \xleftarrow{\langle T_0:-21, p\rangle} E$, $Z \xleftarrow{\langle T_5:-22, \neg p\rangle} E$, $Z \xleftarrow{\langle T_0T_1:-22, p\rangle} E$, $Z \xleftarrow{\langle T_5:-23, p\rangle} E$, $Z \xleftarrow{\langle T_1T_5:-24, p\rangle} E$, $Z \xleftarrow{\langle T_0T_5:-26, \neg p\rangle} E$, $Z \xleftarrow{\langle T_0T_5:-27, p\rangle} E$, $Z \xleftarrow{\langle T_0T_1T5:-28, p\rangle} E$
Upper bounds	$Z \xrightarrow{\langle 86, \neg p\rangle} E$, $Z \xrightarrow{\langle 98, p\rangle} E$

The considered DC checking algorithm on S_P [14] returns a positive answer and a modified S_P. The returned S_P is a *distance graph*.[2] containing the minimal ordinary constraints between Z and any other timepoint (i.e., the minimal and the maximal temporal distance of all reachable timepoints from Z), and possible new constraints, *conjunct-lower/upper-case edges* (such kinds of constraint are discussed in Section 2), between Z and some timepoint, representing the minimal and the maximal temporal distance of such timepoints from Z according to the minimum/maximum duration of one or more guarded links. Therefore, considering Z and the ending point E of S_P, the determined constraints between the two nodes represent the minimum and maximum durations of the process P according to different contingencies/conditions.

Example 4. Considering the FTNU S_{P_2} represented in Fig. 2(a), the DC checking algorithm derives the constraints between Z and E in the corresponding FTNU distance graph, as depicted in Table 4.

The ordinary edge $Z \xleftarrow{\langle -16, \neg p\rangle} E$ represents the minimal duration, i.e., 16 time units, of P in all scenarios where $\neg p$ is $\top$, and all requirement/guarded links last their lower bound, i.e., the value x in $X_i \xrightarrow{[x,y],\,\ell} X_j$/ $A_i \xrightarrow{[[x,x'][y',y]]} C_i$, where $\neg p$ entails ℓ. The ordinary edge $Z \xleftarrow{\langle -17, p\rangle} E$ states that the analogous minimal duration of P is 17 when p is true, i.e., in the two scenarios pq and $p\neg q$.

The ordinary edge $Z \xrightarrow{\langle 98, p\rangle} E$ represents the upper bound to the duration of P in scenarios where p is $\top$ while $Z \xrightarrow{\langle 86, \neg p\rangle} E$ is the upper bound in scenarios where $\neg p$ is $\top$. Such maximum values, 86 and 98, are the maximum durations among all possible executions when the delays among timepoints and the duration of guarded links are maximum.

Regarding conjunct-lower-case constraints, each value represents the upper bound to the minimal process duration when one or more contingent links last their lower-guard duration, i.e., the x' value. As an example, the value $\langle t_0t_1t_2{:}90, p\rangle$ represents the fact that in the case of the Hypoxemic scenario, if the task T_0, T_1 and T_2 last their lower-guard duration (i.e., 4, 1, and 1, respectively), then P can be forced to last at least 90 time-units without constraints violations. If P is forced to last at least 91 time-units by a further constraint, for sure at least one execution will violate such a constraint.

As regards the conjunct-upper-case constraints, each value represents the lower bound to the maximal process duration that one can force without constraint violations when one or more contingent links last their upper-guard duration, i.e., the y' value. As an example, the value $\langle T_0T_1T_5{:}{-28}, p\rangle$ represents the fact that in the case of the *Hypoxemic* scenario, if the task T_0, T_1 and T_5 last their upper-guard duration (i.e., 6, 2, and 16, respectively), then P can be forced to last 28 time-units at most without constraint violations.

Since we want to offer the possibility to consider a process P as a module that can be inserted into other processes, it is important to represent the temporal behavior of a process compactly to make easier the evaluation of whether it can be added to a parent process without violating any temporal constraint of P or of the host process.

[2] A distance graph is an equivalent representation of an FTNU, suitable for having the possible distances between pairs of nodes, as detailed in Section 2.

We propose to represent the temporal behavior of a process using a *prototypal* link between its starting timepoint and the ending one.

Definition 2 (*Prototypal Link*)**.** Given two timepoints A and B, a *prototypal* link is a generalization of a guarded link where the two bounds, x and y, are limited by two *guards*, x' and y'. It is represented as $A \xRightarrow{[\![x, x'][y', y]\!]} B$, where $x, y \in \mathbb{R}$ are the (external) bounds, $x', y' \in \mathbb{R}$ are the *guards*, and it holds that $x \le y$, $x \le x'$, and $y' \le y$.

A prototypal link has no label because its goal is to represent the temporal behavior of the considered child process viewed as a black box where only the starting and the ending timepoint of the subprocess are known. The value x' represents the maximum lower-bound that can be set between A and B without violating any other process constraint. In contrast, y' represents the minimum upper-bound that can be set without violating any constraint. A prototypal link generalizes a guarded link because x' can be greater than y', and timepoint B has not to be a contingent timepoint.

A prototypal link cannot be used directly in an FTNU as a constraint. Instead, it is only useful for describing the global temporal behavior of an FTNU compactly. In the following section, after introducing the contingency span, we will show how a prototypal link can be converted into a proper constraint for being used in an FTNU.

A process P may have different execution scenarios, each with different temporal durations according to (possible) guarded link durations. The prototypal link between its starting timepoint Z and ending one E, $\mathrm{Z} \xRightarrow{[\![x, x'][y', y]\!]} E$, can be determined considering the values among the existing edge values between Z and E in the corresponding FTNU distance graph after a successful DC checking. We propose to set the prototypal link values as follows:

$$\begin{aligned}
x &= \min\{|v| : \mathrm{Z} \xleftarrow{\langle v,\, \ell\rangle} E\}\\
x' &= \min\{v : \mathrm{Z} \xrightarrow{\langle \daleth : v,\, \ell\rangle} E,\ \mathrm{Z} \xrightarrow{\langle v,\, \ell\rangle} E\}\\
y' &= \max\{|v| : \mathrm{Z} \xleftarrow{\langle \aleph : v,\, \ell\rangle} E,\ \mathrm{Z} \xleftarrow{\langle v,\, \ell\rangle} E\}\\
y &= \max\{v : \mathrm{Z} \xrightarrow{\langle v,\, \ell\rangle} E\}
\end{aligned}$$

The lower bound x is the minimum absolute value among the ordinary edges $\mathrm{Z} \xleftarrow{\langle v,\, \ell\rangle} E$, for any value of ℓ, because it represents the minimum possible duration of P in any possible execution. Analogously, the upper bound y is the maximum value among the ordinary edges $\mathrm{Z} \xrightarrow{\langle v,\, \ell\rangle} E$ because it represents the maximum possible duration of P in any possible execution.

If S_P has one or more conjunct-lower-case edges $\mathrm{Z} \xrightarrow{\langle \daleth : v,\, \ell\rangle} E$ (discussed in Section 2), then the minimum duration x cannot be forced to be greater than the minimum of the determined conjunct-lower-case values. This can be represented by setting the lower-guard x' to such a minimum value. However, this is not sufficient. Indeed, since there could exist different scenarios in P having non-overlapping duration ranges like $\langle [l, u], p\rangle$ and $\langle [l', u'], \neg p\rangle$ with $u < l'$, it is necessary also to guarantee that the lower bound x cannot be forced to be greater than the upper bound of any possible execution. In the above example, $x' \le \min\{u, u'\}$. Hence, the lower-guard x' has to be set to the minimum among the conjunct-lower-case values and the upper bounds of the duration of the process, i.e., $x' = \min\{v : \mathrm{Z} \xrightarrow{\langle \daleth : v,\, \ell\rangle} E,\ \mathrm{Z} \xrightarrow{\langle v,\, \ell\rangle} E\}$.

Analogous analysis can be done for the upper-guard y' setting.

Example 5. Moving to the child process P_2, let us consider the constraints determined by S_{P_2} DC checking as depicted in Table 4. The temporal behavior of S_{P_2} can be represented by the prototypal link $\mathrm{Z} \xRightarrow{[\![16, 79][28, 98]\!]} E$. This prototypal link means that it is possible to "force" some restriction of process duration like adding a new constraint $\mathrm{Z} \xrightarrow{[30, 98],\, \boxdot} E$, but restrictions like $\mathrm{Z} \xrightarrow{[16, 25],\, \boxdot} E$, for example, are not allowed.

Definition 3 (*Lower and Upper Guards of Timepoint*)**.** In general, given an FTNU S and the prototypal link $\mathrm{Z} \xRightarrow{[\![x, x'][y', y]\!]} B$ determined as previously described, we denote x' also as $\mathrm{lG}_S(B)$ calling it *lower guard of timepoint B in S*, and the y' as $\mathrm{uG}_S(B)$ calling it *upper guard of B in S*.

This representation of the global temporal behavior of a process as a prototypal link is a generalization of the proposal made in [9]. In [9], authors proposed a definition of lower-guard and upper-guard for a process without conditions and, then, an algorithm that determines such values making a breath-first visit of the corresponding graph. Here, we generalize the definition of lower-/upper-guard, considering all possible process scenarios and exploiting the results obtained by the DC checking algorithm for determining the values without further computation.

Example 6. Considering the child process P_6, whose FTNU S_{P_6} is in Fig. 2(b), the DC checking algorithm determines the following new constraints (compact representation) in the FTNU distance graph associated with S_{P_6}:
$\mathrm{Z} \xleftarrow{\langle -2,\, \boxdot\rangle,\ \langle T_0 : -6,\, \boxdot\rangle,\ \langle T_1 : -6,\, \boxdot\rangle,\ \langle T_0T_1 : -10,\, \boxdot\rangle} E$, and
$\mathrm{Z} \xrightarrow{\langle 20,\, \boxdot\rangle,\ \langle t_0t_1 : 8,\, \boxdot\rangle,\ \langle t_0 : 14,\, \boxdot\rangle,\ \langle t_1 : 17,\, \boxdot\rangle} E$.

Therefore, the prototypal link representing the P_6 temporal behavior is $\mathrm{Z} \xRightarrow{[\![2, 8][10, 20]\!]} E$.

5.3. Contingency span

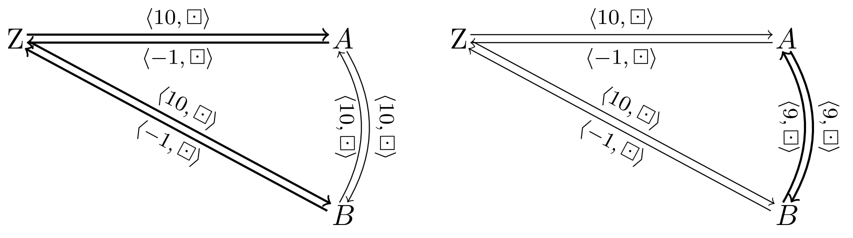
Given the range $[\![x, x'][y', y]\!]$ of a prototypal link associated with a DC process, if we want to apply a restriction on the duration of the process, we can limit the lower/upper bound or both. While limiting one bound (lower or upper) only according to its guards always does not affect the dynamic controllability of the process, limiting both bounds can raise an over-constrained situation, i.e., the process may result in no more DC.

Example 7. Consider the FTNU from Fig. 2(b), which corresponds to child process P_6. In the previous section, we have shown that the global temporal behavior can be represented by the prototypal link $\mathrm{Z} \xRightarrow{[\![2, 8][10, 20]\!]} E$. If the duration is forced to be 8 at the minimum and 10 at the maximum adding the requirement link $\mathrm{Z} \xrightarrow{[8, 10],\, \boxdot} E$ ($8 = \mathrm{lG}_{P_6}(E)$, $10 = \mathrm{uG}_{P_6}(E)$), then the process is no more DC. Indeed, the guarded link between timepoints T_{0_S} and T_{0_E} (representing task T_0) has a duration variability of 5 time-units (i.e., from 1 to 5), which can be "mitigated" by the following delay by only one time-unit. Therefore, even considering only T_0, it is impossible to require that the global duration of the overall process has to vary in a range of 2 time-units only.

Thus, as proposed in [9], it is necessary to consider the *contingency span* of the process, an additional value that represents the minimal span to be guaranteed for the duration range. In [9], contingency span was defined for temporal constraint networks without conditions.

Here, we extend the above concept *contingency span* and its correlated ones, i.e., *link contingency span*, and *path contingency span*, for applying to FTNUs.

Such concepts are meaningful only for networks that are (dynamically) controllable, i.e., all constraints can be satisfied during any possible execution. Moreover, the determination of the contingency span of timepoints (or of a network) requires restricting the constraint ranges to contain only admissible values. Indeed, controllable FTNUs processed by the proposed DC-checking algorithm have already derived such restricted constraints between



(a) A DC FTNU (represented as distance graph) where only constraint to/from Z are minimised (thick ones).

(b) Complete DC FTNU (distance graph) of Figure 3a (thick edges are the ones minimised propagating the other ones).

Fig. 3. Exampe of a simple DC FTNU and its corresponding Complete DC FTNU.

Z and any timepoint and between any activation and the corresponding contingent timepoint, but not between any possible pair of timepoints. Thus, we need to update the constraints between any pairs of timepoints suitably, propagating the restricted constraints of such timepoints from/to Z. To this end, we introduce the concept of *complete FTNUs*.

Definition 4 (*Complete DC FTNU*)**.** Given a DC FTNU $S = (\mathcal{T}, \mathcal{P}, \mathcal{OT}, O, L, \mathcal{C}, \mathcal{G})$, its corresponding *complete* DC FTNU $S' = (\mathcal{T}, \mathcal{P}, \mathcal{OT}, O, L, \mathcal{C}', \mathcal{G}')$ contains the minimized version of guarded and requirement links, i.e., $\mathcal{C}'$ contains requirement links of $\mathcal{C}'$, each with the minimal range each, while $\mathcal{G}'$ contains the guarded links of $\mathcal{G}$, each with the minimal guarded range. A minimal (guarded) range is composed by all and only the admissible values guaranteeing the FTNU DC.

Taking into account the FTNU derived by our DC check algorithm, the corresponding complete DC FTNU can be obtained by propagating the minimal links between each timepoint X_i and Z to each X_i-neighbor X_j i.e., a pair (X_i, X_j), for which there exists a requirement link in $\mathcal{C}$.

Example 8. Fig. 3 depicts a DC FTNU (represented as a distance graph) where the constraints between A and B are not minimized on the left and the corresponding Complete DC FTNU on the right, where the constraints between A and B are minimized considering the minimum constraints from/to Z.

The determination of all minimum constraints can be performed in $O(|\mathcal{T}|^2 2^{|\mathcal{P}|})$ time in the worst case because for each timepoint ($|\mathcal{T}|$), all possible neighbors must be considered ($|\mathcal{T}|$) and each edge can have up to $2^{|\mathcal{P}|}$ labeled values.

We are now ready to introduce the concept of contingency span.

Definition 5 (*Labeled Link Contingency Span*)**.** A positive labeled link contingency span Δ is the span that needs to be guaranteed for a guarded link to ensure the DC of an FTNU. A negative labeled link contingency span Δ is the maximum span provided by a link that can be used to reduce the contingency span of an earlier guarded link. They are defined as follows:

(a) Given $A\xrightarrow{[[a,a'][b',b]]}B$, the link contingency span $\Delta_{AB_{\boxdot}}$ is defined as $\Delta_{AB_{\boxdot}} = b' - a'$.
(b) Given $A\xrightarrow{[a,b],\,\ell}B$, the link contingency span Δ_{AB_ℓ} is defined as $\Delta_{AB_\ell} = a - b$.

$\Delta_{AB_{\boxdot}}$ relative to a guarded link is always nonnegative and always has the empty label ($\boxdot$) because guarded links do not have labels.

Δ_{AB_ℓ} relative to a requirement link is usually negative and has the same label (ℓ) as the associated requirement link.

The contingency span of a path is based on the labeled link contingency span of its links, and it is defined only when all links have consistent labels.

In particular, when a guarded link $A\xrightarrow{[[a,a'][b',b]]}B$ is followed by a requirement link $B\xrightarrow{[c,d],\,\ell}C$, then the contingency span required by the guarded link can be partially or fully compensated by the following requirement link, as the latter's duration can be decided based on the actual duration of the former. Thus, the contingency of the path from A to C is given by $\Delta_{AB_{\boxdot}} + \Delta_{BC_\ell}$.

When a requirement link $A\xrightarrow{[a,b],\,\ell}B$ is followed by a guarded link $B\xrightarrow{[[c,c'][d',d]]}C$, then the requirement link cannot be used to compensate for the contingency span of the guarded link because the duration of the requirement link has to be decided before executing the guarded link. Hence, the contingency span of the path from A to C is given by $\Delta_{BC_{\boxdot}}$.

The contingency of the sequence of two guarded links is the sum of the contingency spans of the two guarded links.

The contingency of the sequence of two requirement links, $A\xrightarrow{[c,d],\,\alpha}B$, $B\xrightarrow{[c',d'],\,\beta}C$, is the contingency $\Delta_{AC_\gamma} = \Delta_{AB_\alpha} + \Delta_{BC_\beta}$ when $\gamma = \alpha\beta$ is consistent, i.e., $\gamma \in \mathcal{P}^*$. When γ is not consistent, the contingency span of the path does not exist because the two links belong to different scenarios that cannot occur in the same execution.

For example, suppose $A\xrightarrow{[c,d],\,p}B$ and $B\xrightarrow{[c',d'],\,\neg p}C$ belong to an FTNU that has the three conditions p, q, and r. In that case, the first constraint must be considered in four of eight scenarios, i.e., $pqr, pq\neg r, p\neg qr, p\neg q\neg r$, while the second constraint in the other four scenarios, i.e., $\neg pqr, \neg pq\neg r, \neg p\neg qr, \neg p\neg q\neg r$. There is no scenario in which the two constraints have to be both considered; therefore, no significant path can be made using the two constraints.

Finally, since there may be different requirement links (having different labeled values) between two timepoints, the contingency span of a path involving the two considered timepoints (and requirement links) may be defined as a set of different labeled values.

Example 9. Consider the guarded link $A\xrightarrow{[[10,12][16,24]]}B$ followed by two different requirement links to timepoint C, $B\xrightarrow{[0,2],\,p}C$ and $B\xrightarrow{[0,10],\,\neg p}C$. In scenarios where p is $\top$, C has to occur within 2 time-units after B, while in scenarios where p is $\perp$, C has to occur within 10 units after B. In scenarios where p is $\top$, the contingency span of the path (A, B, C) is $\Delta_{AC_p} = (16-12)+(0-2) = 2$, while in scenarios where p is $\perp$, $\Delta_{AC_{\neg p}} = (16-12)+(0-10) = -6$.

A path of two or more links connecting two nodes determines an implicit constraint between the two nodes. Such a constraint can also be explicitly represented (usually as a requirement link) or not. Given a path, the labeled link contingency spans that can be defined in such a path need to be combined incrementally to determine the labeled contingency span between the first timepoint and the last one. It is meaningful to consider Z as the first timepoint. Since there may be more paths connecting Z to a timepoint X, more distinct labeled contingency spans are possible. The largest span among such labeled contingency spans represents the most demanding contingency span that can occur during executions. This maximum labeled contingency span is called *path contingency span* of X, which is an extension of the path-contingency-span concept presented in [9]. Here, we redefine it because it depends on the labeled link contingency span concept.

Definition 6 (*Labeled Path Contingency Span*). Let S be a DC complete FTNU. The *labeled path contingency span* of Z is $\mathrm{cont}_S(\mathrm{Z}_{\boxdot}) = 0$. The *labeled path contingency span* $cont_S(C_\ell)$ of any other timepoint C in scenario ℓ is defined as

$$\mathrm{cont}_S(C_\ell) := \max\left\{0, \max_{B\in\mathcal{T}}\left\{\mathrm{cont}_S(B_\alpha) + \Delta_{BC_\beta} \mid \alpha\beta \text{ are consistent with } \ell\right\}\right\}$$

Once all possible $\mathrm{cont}_S(C_\ell)$ are determined, we define as *path contingency span* of timepoint C the value

$$cont_S(C) := \max_{\ell\in\mathcal{P}^*}\{\mathrm{cont}_S(C_\ell)\}$$

The labeled path contingency span of any timepoint is always greater or equal to zero, i.e., $\mathrm{cont}_S(C) \geq 0$. Thus, the problem of determining the value of $\mathrm{cont}_S(C)$ can be reduced to the problem of finding the minimal distance(s) between Z and C in a labeled weighted graph where each edge (A, B) has a set of values given by the negated labeled link contingency spans $-\Delta_{AB_\ell}$ of the edge (A, B) in the network.

Definition 7 (*Labeled Contingency Graph*). Let $S = (\mathcal{T}, \mathcal{P}, \mathcal{OT}, O, L, \mathcal{C}, \mathcal{G})$ be a DC complete FTNU. The corresponding *labeled Contingency Graph* for S has the form $\mathcal{CG}_S = (\mathcal{T}, \mathcal{E}_{\mathcal{CG}_S})$, where the nodes are the same of S and $\mathcal{E}_{\mathcal{CG}_S}$ is a set of weighted edges having:

(a) For each guarded link $A \xrightarrow{[[x,x'][y',y]]} B \in \mathcal{G}$, a single edge $A \xrightarrow{\langle -\Delta_{AB_{\boxdot}}, \boxdot\rangle} B$.
(b) For each requirement link $A \xrightarrow{[x,y],\alpha} B \in \mathcal{C}$, two edges, $A \xrightarrow{\langle -\Delta_{AB_\alpha}, \alpha\rangle} B$, and $B \xrightarrow{\langle -\Delta_{AB_\alpha}, \alpha\rangle} A$.
(c) For each timepoint $T \in \mathcal{T}$, a single edge $\mathrm{Z} \xrightarrow{\langle 0, \boxdot\rangle} T$.

It is straightforward to observe that, according to this definition, the determination of a labeled contingency graph requires a $O(|\mathcal{T}| + |\mathcal{C}| + |\mathcal{G}|)$ time.

In a labeled Contingency Graph, a labeled path is a sequence $(X_0, e_0, X_1, e_1, \ldots, X_k)$, where

(i) $X_i \in \mathcal{T}$ for $i = 0, \ldots, k$,
(ii) $e_j \in \mathcal{E}_{\mathcal{CG}_S}$ for $j = 0, \ldots, k-1$,
(iii) the conjunction of all propositional labels associated with the considered edges is consistent, i.e., $\ell_0\ell_1 \cdots \ell_{k-1} \in \mathcal{P}^*$, where ℓ_j is the label associated with e_j, for $j = 0, \ldots, k-1$.

The *path contingency span* of any timepoint $X \in \mathcal{T}$ in scenario ℓ corresponds to the negative value of the shortest path from the initial timepoint Z to X in the corresponding *labeled contingency graph* considering only edge values having labels consistent with ℓ.

Once all possible path contingency spans of a timepoint X have been determined, the minimum value among such path contingency spans represents the negated value of the labeled path contingency span of X.

Since a requirement link may connect two-non sequential timepoints, its link contingency span can be combined with

Algorithm 1: ContingencyDistances($\mathcal{CG}_S$)

```
Input: $\mathcal{CG}_S = (\mathcal{T}, \mathcal{E}_{\mathcal{CG}_S})$, the labeled Contingency Graph associated with a DC FTNU $S$
Output: $[\mathrm{cont}_S(X_1), \mathrm{cont}_S(X_2), \ldots, \mathrm{cont}_S(X_n)]$, the path contingency span of each node in $\mathcal{CG}_S$
1  foreach $v \in \mathcal{T}$ do                                   // Initialization
2      $v.d := \{\langle\infty, \boxdot\rangle\}$          // Each node has a set of labeled distances
3  $Z.d := \{\langle 0, \boxdot\rangle\}$                                      // $Z \equiv X_0$
4  foreach $1 \le i \le |\mathcal{T}| - 1$ do                   // Distance propagation
5      foreach $(X_i, \langle e, \alpha\rangle, X_j) \in \mathcal{E}_{\mathcal{CG}_S}$ do
6          foreach $\langle u, \beta\rangle \in X_i.d$ do                        // Relax phase
7              if $\alpha\beta \notin \mathcal{P}^*$ then continue next cycle    // Not consistent label
8              useless := $\bot$
9              foreach $\langle v, \gamma\rangle \in X_j.d$ do     // Check if the new value can be added
10                 if $\alpha\beta \supset \gamma$ and $v \le u + e$ then
                       // $\langle v, \gamma\rangle$ is stricter, $u + e$ is not useful
11                     useless := $\top$
12                     break
13             if useless then continue next cycle
14             $X_j.d := X_j.d \cup \{\langle u + e, \alpha\beta\rangle\}$    // Add the new labeled distance
15             foreach $\langle v, \gamma\rangle \in X_j.d$ do            // Remove looser distances
16                 if $\gamma \supset \alpha\beta$ and $v \ge u + e$ then
17                     $X_j.d := X_j.d \setminus \{\langle v, \gamma\rangle\}$
   /* Negative cycle check is not necessary                                  */
18 $\mathrm{cont}_S := []$                                        // empty solution array
19 foreach $0 \le i \le |\mathcal{T}|$ do
20     $\mathrm{cont}_S[i] := \max\{|v| \mid \langle v, \ell\rangle \in X_i.d\}$
21 return $\mathrm{cont}_S$
```

the contingency coming from any of its endpoints. Therefore, Definition 7 considers these two mutually-exclusive options by adding two edges $A \xrightarrow{\langle -\Delta_{AB_\alpha}, \alpha\rangle} B$, $B \xrightarrow{\langle -\Delta_{AB_\alpha}, \alpha\rangle} A$ in $\mathcal{E}_{\mathcal{CG}_S}$.

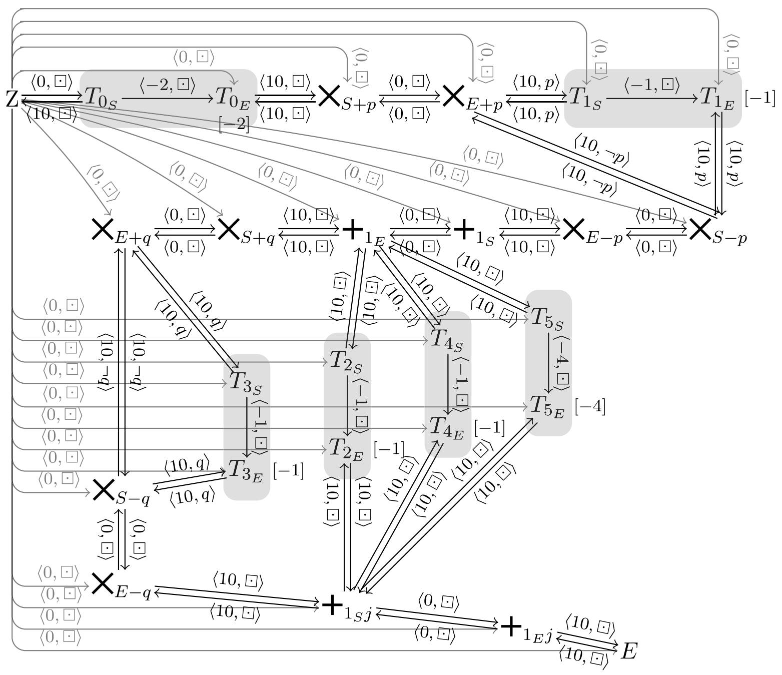
The edges $\mathrm{Z} \xrightarrow{\langle 0, \boxdot\rangle} T$ added in Item (c) of Definition 7 guarantee that the length of any shortest path to a timepoint X from the starting timepoint Z is always non-positive, i.e., the corresponding labeled path contingency is always positive.

Given a DC complete FTNU S, the labeled contingency graph $\mathcal{CG}_S$ cannot contain any negative cycles because a negative cycle in $\mathcal{CG}_S$ could be easily translated into a negative cycle in S, a contradiction because a DC FTNU cannot contain negative cycles.

As an example, Fig. 4 depicts the labeled contingency graphs of the two child processes P_2 and P_6.

As an original contribution, here we propose algorithm `ContingencyDistances` (see Algorithm 1), which calculates the path contingency span $\mathrm{cont}_S(X)$ for each $X \in \mathcal{T}$ of a given labeled contingency graph. `ContingencyDistances` is an adaptation of the Bellman–Ford algorithm to determine the minimum distances of all nodes from a single source node [27].

After initialization (Lines 1 to 3), the algorithm executes for $|\mathcal{T}|-1$ time the relaxation phase (Lines 6 to 17) for each edge [27, page 648]. Since each node can have more distances (already determined) from Z, each having a different propositional label, the "relax" action of an edge must be done for each possible distance from Z of the source node of the edge (**foreach** on line 6). Each combination of source node distance and edge labeled value determines a possible new distance for the destination node of the edge when the propositional labels are consistent (Line 7). Once a new distance for the destination node is possible, it has to be compared with all stored distances of the destination node. If any stored distance of the destination node is stricter than the new one, then the new one is discarded, and the algorithm jumps to the next distance of the source node (Lines 9 to 13). For example, if the new distance is $\langle -3, pq\rangle$ and X_j has distance $\langle -4, p\rangle$, then $\langle -3, pq\rangle$ is useless because in scenarios where p is $\top$, the distance has to be 4 at minimum. If the new distance has not been discarded, then it is added, and all already


(a) Contingency Graph corresponding to P_2.

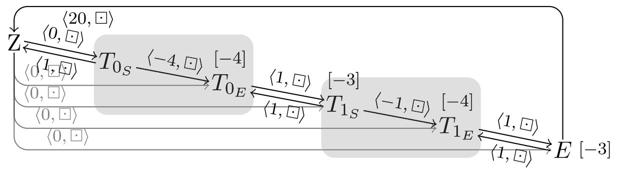

(b) Contingency Graph corresponding to P_6.

Fig. 4. Contingency graph of the FTNUs in Fig. 2 showing values determined by the `ContingencyDistances`. For clarity, only non-0 contingency span values are shown (node bracketed values).

stored distances are checked if they are looser than the new one (Lines 14 to 17). For example, if the new distance is $\langle -3, pq\rangle$ and X_j has $\langle -2, pqr\rangle$ as a stored distance, then $\langle -2, pqr\rangle$ is useless because, in scenarios where pq is $\top$, the distance has to be three at minimum. After the main cycle is finished, all nodes contain the labeled minimal distances from Z. The path contingency span of a node is the maximum absolute value of such distances (Lines 18 to 21). The algorithm executes $|\mathcal{T}|$ main cycles. In each main cycle, it *relaxes* each edge ($|\mathcal{E}_{\mathcal{CG}_S}|$ relax phases); since each timepoint can have up to 2^k different distances in the worst case, each "relax" phase requires $O(2^k)$ time, where k is the number of observation timepoints in the FTNU S. Therefore, the algorithm requires $O(2^k|\mathcal{T}||\mathcal{E}_{\mathcal{CG}_S}|)$ time to determine all minimal distances.

Example 10. Fig. 4 shows the two path contingency graphs corresponding to the FTNUs depicted in Fig. 2. Applying the `ContingencyDistances` to each graph, the contingency span values are determined for each node. The figure shows only non-0 contingency span values in brackets near their corresponding nodes.

The result is that

- $\mathrm{cont}_{S_{P_2}}(E) = 0$, and
- $\mathrm{cont}_{S_{P_6}}(E) = 3$.

It confirms our observation in Example 7 about the impossibility of setting a constraint with a duration variability of just two time-units for P_6.

Based on Definition 6, it becomes possible to describe the admissible duration ranges between two timepoints in an FTNU.

Lemma 1 (*Contingency Span Necessity*)**.** *Let S be a DC complete FTNU, Z be its initial timepoint, X be any other timepoint, and $Z\xrightarrow{[x,x'][y',y]}X$ the prototypal link between Z and X determined as shown in Section 5.2.*

To guarantee S to be dynamically controllable for any restriction $Z\xrightarrow{[u^*,v^*]}X$ ($x \le u^* \le x'$, $y' \le v^* \le y$) of the distance between Z and X, it has to hold that $v^* - u^* \ge \mathrm{cont}_S(X)$ holds.

Now, we prove some important relationships between the $\mathrm{lG}_S(X)$, $\mathrm{uG}_S(X)$, and $\mathrm{cont}_S(X)$ values:

Lemma 2 (*Contingency Span Sufficiency for Upper Guard*)**.** *Let S be a complete DC FTNU, X be a timepoint, and $Z\xrightarrow{[x,\,\mathrm{lG}_S(X)][\mathrm{uG}_S(X),\,y]}X$ the prototypal link between Z and X. Let T a copy of S where the lower bound of the distance between Z and X is set to the value u^*, where $x \le u^* \le \mathrm{lG}_S(X)$. Then, the prototypal link between Z and X in T becomes $Z\xrightarrow{[u^*,\,\mathrm{lG}_S(X)][\mathrm{uG}_T(X),\,y]}X$, where $\mathrm{uG}_T(X) = \min\{\mathrm{uG}_S(X), u^* + \mathrm{cont}_S(X)\}$ to guarantee that any constraint derived from it can be satisfied in T.*

Lemma 3 (*Contingency Span Sufficiency for Lower Guard*)**.** *Let S be a complete DC FTNU, X be a timepoint, and $Z\xrightarrow{[x,\,\mathrm{lG}_S(X)][\mathrm{uG}_S(X),\,y]}X$ be the prototypal link between Z and X. Let T a copy of S where the upper bound of the distance between Z and X is set to the value v^*, where $\mathrm{uG}_S(X) \le v^* < y$. Then, the prototypal link between*

Z and X in T becomes $Z\xRightarrow{[\![x,\, \mathrm{lG}_T(X)][\mathrm{uG}_S(X),\, v^*]\!]}X$, *where* $\mathrm{lG}_T(X) = \min\{\mathrm{lG}_S(X), v^* - \mathrm{cont}_S(X)\}$ *to guarantee that any constraint derived from it can be satisfied in T.*

5.4. Prototypal Links with Contingency (PLCs) for the overall temporal representation of a process

Now it is possible to give a complete description of how a prototypal link representing the overall temporal properties of a process can be used.

Let us start by introducing the concept of *prototypal link with contingency*.

Definition 8 (*Prototypal Link with Contingency (PLC)*)**.** Let Z and E be the single initial timepoint and the single end timepoint of an FTNU S, respectively.

A *prototypal link with contingency (PLC)* between Z and E is an extension of a prototypal link, where the two bounds x and y are limited by two guards, x' and y', and by a contingency span c. It is represented as $Z\xRightarrow{[\![x, x'][y', y]\!]\updownarrow c}E$, where $x, y \in \mathbb{R}$ are the (external) bounds, $x', y' \in \mathbb{R}$ are the *guards*, c is a contingency span, and it holds that $x \le y$, $x \le x'$, and $y' \le y$.

A PLC $Z\xRightarrow{[\![x, x'][y', y]\!]\updownarrow c}E$ may correspond to any requirement link with a duration in a range $[x^*, y^*]$ between Z and E, i.e., $Z\xrightarrow{[x^*, y^*],\, \boxdot}E$, with $x^* \le y^*$, where $x \le x^* \le x'$, $y' \le y^* \le y$, and $y^* - x^* \ge c$.

Theorem 2 (*Overall Temporal Properties of a Process*)**.** *Consider a dynamically-controllable process P and its corresponding FTNU S_P (completed with all the constraints determined by the DC checking algorithm). Let Z and E be the single initial timepoint and the single end timepoint of S_P, respectively. The* overall temporal properties *of P can be described by a PLC* $Z\xRightarrow{[\![x, x'][y', y]\!]\updownarrow c}E$, *where*

- *x is the minimal bound of the all possible* $Z\xrightarrow{[x, y],\, \ell}E$ *between Z and E in S derived by the DC checking algorithm,*
- *y is the maximal bound of the all possible* $Z\xrightarrow{[x, y],\, \ell}E$ *between Z and E in S derived by the DC checking algorithm,*
- $x' = \mathrm{lG}_S(E) = \min\{v : Z\xrightarrow{\langle \daleth: v,\, \ell\rangle}E, Z\xrightarrow{\langle v,\, \ell\rangle}E\}$
- $y' = \mathrm{uG}_S(E) = \max\{|v| : Z\xleftarrow{\langle \aleph: v,\, \ell\rangle}E, Z\xleftarrow{\langle v,\, \ell\rangle}E\}$, *and*
- $c = \mathrm{cont}_S(E)$.

Proof. The overall temporal properties of a process allow one to decide how to constrain the duration range of a process without making the process not dynamically controllable.

In particular, given a PLC $Z\xRightarrow{[\![x, x'][y', y]\!]\updownarrow c}E$ of a process P,

- **it is possible to restrict the duration range of** P in a range $[u^*, v^*]$ setting a requirement link between Z and E, i.e., $Z\xrightarrow{[u^*, v^*],\, \boxdot}E$ observing that $u^* \le v^*$, $x \le u^* \le x'$ and $y' \le v^* \le y$, and that $v^* - u^* \ge c$ to preserve the DC of P.
- **it is possible to represent** P **inside another process** M as a black box having a duration in a range $[u^*, v^*]$, setting a guarded link $X\xRightarrow{[\![x, u^*][v^*, y]\!]}Y$ in M between two timepoint X and Y representing the starting timepoint and the ending timepoint of P in M observing that $u^* \le v^*$, $x \le u^* \le x'$ and $y' \le v^* \le y$, and that $v^* - u^* \ge c$ to preserve the DC of P.

Lemmas 1 to 3 show how to use the path contingency span $\mathrm{cont}_S(E) = c$ to ensure that any possible restriction of the duration range $[\![x, x'][y', y]\!]$ of the process preserves its DC. □

Example 11. Considering the child process P_2 in Fig. 1 and its corresponding FTNU in Fig. 2, the PLC range may describe its overall temporal properties $[\![16, 79][28, 98]\!] \updownarrow 0$. Therefore, it is possible to restrict the overall duration range of the process to $[16, 28]$ or $[30, 30]$, for example, while still preserving its DC. For process P_6 (cf. Figs. 1 and 2), its temporal behavior can be described by the PLC range $[\![2, 8][10, 20]\!] \updownarrow 3$. Therefore, for example, the duration range of the process can be restricted to $[7, 10]$, or $[8, 11]$, while it is not possible to restrict to $[8, 10]$ as already shown in the previous section.

6. Checking the dynamic controllability of modularized time-aware processes

As previously addressed in [9] for a simple fragment of BPMN constructs, and now suitably extended to deal with possibly alternative execution paths, we can represent each (child-)process by a *prototypal link with contingency (PLC)*. In particular, a PLC specifies the allowed spans of the process as well as the permissible restrictions that may be applied without violating the DC of the process.

In this section, we propose and discuss new aspects, focusing on

- how we can use such PLCs to manage the temporal properties of a parent process, in the presence of (possibly) many child processes taken from a library. Indeed, not all the possible guarded ranges, compatible with PLCs for different child processes, are fine concerning the controllability of the parent process.
- different kinds of DC violations. Indeed, it could be that (i) a child process exceeds the maximum allowed duration inside the parent process, or (ii) a child process is too fast concerning the allowed time within the parent process. While we can do nothing to solve the first kind of DC violation, in the second case, we can restore the DC if we were allowed to "wait" during the execution of the parent process.
- a different way of managing the second kind of DC violation modifying the child process specification. Intuitively, we may put the above-mentioned "wait" inside the child process to make it even more flexible and, thus, reusable.

6.1. Configuring child processes

The following approach, when a designer has to build a parent process using a repository of process schemata, is similar to the one proposed in [9].

First, such a repository contains processes (each with schemata and possibly instantiations) that a designer can use in a parent process as child processes (BPMN subprocesses). Each process has a PLC representing its temporal behavior.

Second, when a designer wants to use a child process P_i, he/she has to select a guarded range allowed by the P_i PLC and sets it as temporal property of P_i in the parent process. In this way, P_i in the parent process is represented as a guarded link that can be only partially modified by the parent process during runtime.

However, such a "selecting guarded range" task can be long and tedious because not all the guarded ranges allowed by a PLC for a child process may be compatible with the guarded ranges of all other child processes/tasks and with all possible temporal constraints in the parent process.

Example 12. Fig. 5 depicts the modularized process from Fig. 1.

Let us consider a simple case involving only child process P_2. A designer can insert the child process P_2 from the repository setting the allowed guarded range $[\![65, 65][65, 65]\!]$ because P_2 PLC allows it. With such a temporal property, P_2 will be DC while the parent process will be not DC. Indeed, P_2 duration will be 65 time-units exactly, and such a value violates the constraint that limits to 60 the maximal span between the start of child

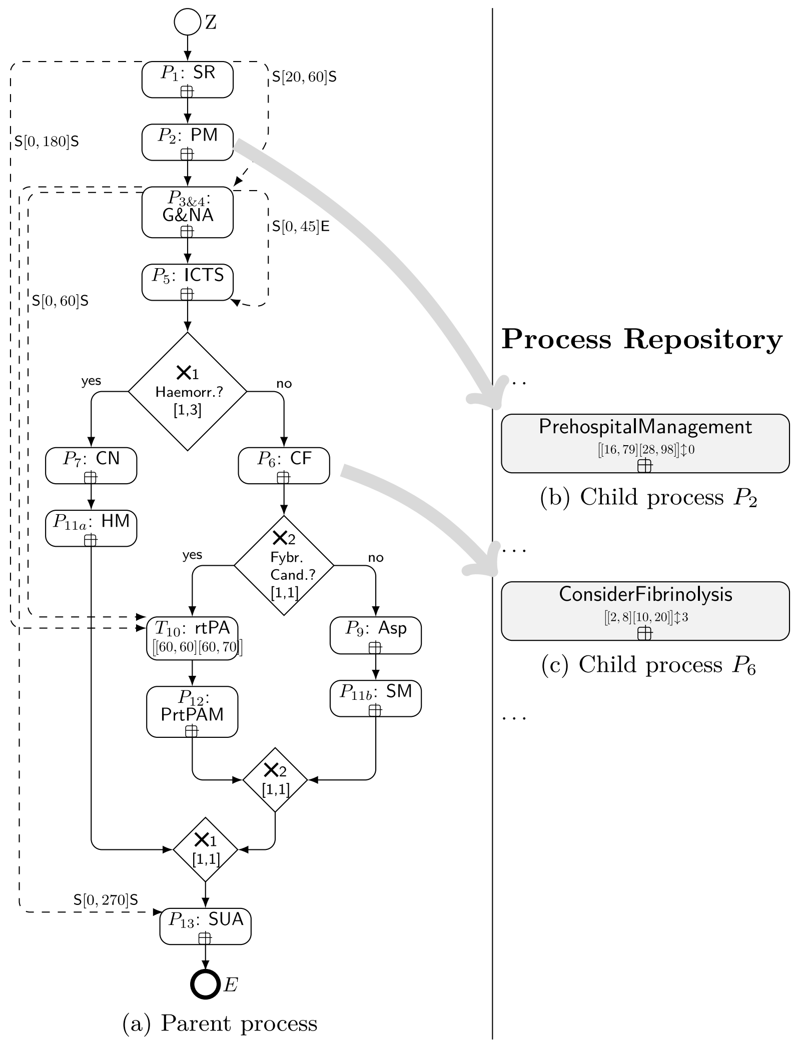


Fig. 5. Modularized process.

process P_1 (predecessor of P_2) and the start of child process $P_{3\&4}$ (successor of P_2). Things become more complicated when the designer has to "insert" child processes $P_1, P_2, P_{3\&4}, P_5$, and P_6 choosing guarded ranges for them that satisfy time lag constraints among such child processes and task T_{10}.

To simplify such a selecting activity, we propose here the novel algorithm `ConfigureSubProcesses` (see Algorithm 2) that, given a parent process, a set of child processes (each described by a PLC), and the number n of desired possible solutions, determines n combinations of guarded ranges for child processes that guarantee the DC of the parent process. If there is no possible solution o fewer than n possible solutions, `ConfigureSubProcesses` returns a diagnostic message.

The algorithm uses the FTNU DC-checking algorithm for checking the DC of the parent process [14]. Since the FTNU DC-checking algorithm does not return details about violated constraints when the given network is not DC, the proposed algorithm finds successful combinations of guarded ranges through an exhaustive search.

The algorithm works only when temporal constraint values are in the integer domain and the temporal unit is 1.

Let M be the parent process, and $P_i, i = 1, \dots, k$, the child processes to use into M. By S_M, we denote the FTNU corresponding to M that has some nodes representing the placeholders where to "insert" the child processes. Each process P_i is DC and has a PLC, $A_i \xrightarrow{[[x_i, x_i'][y_i', y_i]]\updownarrow c_i} C_i$, that describes its temporal behavior. Nodes A_i and C_i are also the name of nodes in S_M where to insert the subprocess P_i.

In general, the set P of P_i, $i = 1, \dots, k$, can be divided into three distinct subsets.

The first set, that we call P^1, is made of P_i having $A_i \xrightarrow{[[x_i, x_i'][y_i', y_i]]\updownarrow c_i} C_i$ with $y_i' - x_i' > c_i$. Such a kind of PLC is possible when P_i has more alternative paths with different contingent constraints on them. For such a kind of P_i, there is only **one** guarded link representing its behavior in a parent process: $A_i \xrightarrow{[[x_i, x_i'][y_i', y_i]]} C_i$. Therefore, the algorithm will insert such a guarded link into the parent network for each P_i of this set.

The second set, that we call P^+, is made of P_i having $A_i \xrightarrow{[[x_i, x_i'][y_i', y_i]]\updownarrow c_i} C_i$ with $c_i > 0$ and $y_i' - x_i' \le c_i$. For such a kind of P_i, there exist different guarded links that represent its behavior in a parent process: $A_i \xrightarrow{[[x_i, b][b + c_i, y_i]]} C_i$ for each integer b such that $x_i \le b \le x_i'$ and $y_i' \le b + c_i \le y_i$. The algorithm will select which of such guarded links allow the controllability of the parent network.

The third set, that we call P^0, is made of P_i having $A_i \xrightarrow{[[x_i, x_i'][y_i', y_i]]\updownarrow c_i} C_i$ with $c_i = 0$. For such a kind of P_i, $y_i' \le x_i'$ and there exist different guarded links that could represent its behavior: $A_i \xrightarrow{[[x_i, b][b, y_i]]} C_i$ for each integer b such that $y_i' \le b \le x_i'$. Each of these guarded constraints represents a constraint where the core consists of only one value, b. This means that the child

Algorithm 2: ConfigureSubProcesses($M, P = [P_1, \ldots, P_k], n$)

```
Input: $M$ is the parent process with placeholders for child processes in
       $P = [P_1, \ldots, P_k]$; $n$ is the number of desired solutions.
Output: $SOL$, set of all possible guarded-range combinations for child
        processes that guarantee the DC of $M$, with $|SOL| \le n$, if $M$ admits
        $SOL$, an error message otherwise.
/* Assumptions: 1) $S_M$ is the FTNU associated to $M$; 2) $A_i, C_i \in S_M$ represent the
   placeholders for the child process $P_i$, for $i = 1, \ldots, k$; $A_i$ is the starting node; $C_i$ is
   the end one; 3) Each $P_i$ is DC and its PLC is $A_i \xRightarrow{[[x_i, x'_i][y'_i, y_i]] \updownarrow c_i} C_i$; 4) Temporal values
   are in the domain of integers.                                                  */
1  $P^+ := \emptyset, P^0 := \emptyset$  // $P^+$ is the set of $P_i$ having $c_i > 0$ that must be managed, $P^0$ is
     the set of $P_i$ having $c_i = 0$
2  foreach $A_i \xRightarrow{[[x_i, x'_i][y'_i, y_i]] \updownarrow c_i} C_i$ associated to $P_i \in P$ do  // Find bounds in $S_M$ for $P_i$
3      if $0 < c_i < y'_i - x'_i$ then              // $x'_i, y'_i$ are relative to child process $P_i$
4          Set and freeze the guarded link $A_i \xRightarrow{[[x_i, x'_i][y'_i, y_i]]} C_i$ in $S_M$  // $P_i$ can be
             represented by only one guarded link. No other possible ranges are possible.
5      else
6          Set the requirement link $A_i \xrightarrow{[x_i, y_i]} C_i$ in $S_M$      // $x_i, y_i$ are relative to $P_i$
7          if $ci > 0$ then Add $P_i$ to $P^+$
8          else Add $P_i$ to $P^0$
9  $dc :=$ FTNU-DC-Check($S_M$)
10 if $\neg dc$ then return "M is not DC even relaxing all bounds of all child
     processes."
/* Let $x_i^M, y_i^M$ the new bounds for the requirement link $A_i \xrightarrow{[x_i, y_i]} C_i$ determined by the
   DC-checking algorithm; DC-checking algorithm guarantees that $-\infty < x_i^M, y_i^M < +\infty$
   */
11 foreach $A_i \xrightarrow{[x_i^M, y_i^M]} C_i \in S_M$ relative to $P_i \in P^+ \cup P^0$ do      // Necessary conditions
12     if $(x_i^M > x'_i) \vee (y_i^M < y'_i) \vee (c_i > 0 \wedge (y_i^M - x_i^M) < c_i)$ then
13         return "$P_i$ cannot be insert into M: the available time-span is
             insufficient."
14     if $x'_i == \infty \vee x'_i > y_i^M$ then $x'_i := y_i^M$
15 foreach $1 \le i \le |P^+|$ do $R_i := \{\}$       // $R_i$ is the ordered set of possible guarded
     ranges for $P_i \in P^+$
16 foreach $A_i \xRightarrow{[[x_i, x'_i][y'_i, y_i]] \updownarrow c_i} C_i$ associated to $P_i \in P^+$ do // Calc. all allowed guarded
     ranges
17     if $x_i < x_i^M$ then $x_i := x_i^M$
18     if $y_i > y_i^M$ then $y_i := y_i^M$
19     foreach $x_i \le b \le x'_i$ do
20         if $y'_i \le b + c_i \le y_i$ then Add $\big[[x_i, b][b + c_i, y_i]\big]$ to $R_i$
21 $R^* := \Pi_{1 \le i \le |P^+|} R_i$ // Cartesian product of all possible guarded ranges of all $P_i \in P^+$
   // Let $R^*_r$ the $r$-th row of $R^*$. $R^*_{r,i} = \big[[x_i^M, x'_i][y'_i, y_i^M]\big]^r$ the guarded range of the $i$-th
      child process in the $r$-th row
22 $SOL := \emptyset$                                              // The set of solutions
23 foreach $r$-th row in $R^*$ do                            // Check each possible combination
24     $S_{M'} :=$ a copy of $S_M$
25     foreach $1 \le i \le |P^+|$ do
26         In $S_{M'}$, replace $A_i \xrightarrow{[x_i^M, y_i^M]} C_i$ with $A_i \xRightarrow{[[x_i^M, x'_i][y'_i, y_i^M]]^r} C_i$ using guarded
             range $R^*_{r,i}$
27     $dc :=$FTNU-DC-Check($S_{M'}$)
28     if $dc \wedge$ for each $P_i \in P^0$, $x_i^M, y_i^M$ in $S_{M'}$ satisfy $x_i^M \le x'_i$ and $y_i^M \ge y'_i$
         then
29         Add the $r$-th row of $S^*$ to $SOL$
30         if $|SOL| == n$ then break
31 if $|SOL| == 0$ then return "No child process configuration possible."
32 if $|SOL| < n$ then return "M admits fewer solutions:", $SOL$
33 return $SOL$
```

process is guaranteed to run without error also using b time unit exactly. Therefore, it is not necessary to check each of such guarded ranges for different values of b against all those of the other threads because it is possible to decide at run-time which one to select. It is sufficient to verify that the outer bounds respect the guards, i.e., $x_i \le x'_i$ and $y_i \le y'_i$ for each $P_i \in P^0$, after the setting of all other guarded-ranges relative $P_i \in P^+$ and a controllability check of all parent process because only in this case it is guarantee to select a suitable b at run-time.

Considering such analysis, the algorithm has been subdivided into different and successive phases.

In the first phase, the algorithm checks if the given child processes can be embedded into the parent process without not-admissible restrictions on the ranges. In detail, in Lines 1 to 8, the algorithm adds the guarded range associated with child processes in P^1 to the parent process, while for all the others it sets a requirement link for each of them using the outer bound values of the prototypal links. This is necessary for finding which outer bound values the parent process needs to restrict to guarantee controllability. In this phase, it also "subdivides" subprocesses into two different sets, P^+ and P^0.

In Line 9, it checks the DC property of S_M. If the network S_M is not DC, then it means that at least one child process has a minimum or maximum duration that violates some constraints in the parent process.

For example, if a child process P_k has the upper guard equals to 16, but the parent process allows only 15 as the maximum duration, then the network is not DC. In this case, P_k cannot be considered and, therefore, the list P of child processes is incompatible with M; the algorithm has to stop the execution.

If S_M is DC, it contains the new bounds $x_i \le x_i^M, y_i^M \le y_i$ in the requirement link $A_i \xrightarrow{[x_i, y_i]} C_i$; DC-checking algorithm guarantees that $-\infty < x_i^M, y_i^M < +\infty$ because it guarantees that each node has a finite time window. Such bounds are the ones that the DC property of the parent process dictates to child processes.

For example, if a child process P_k has a proper maximal duration of 98, i.e., $y_k = 98$, but the DC checking of the parent process finds that the maximum allowed duration for P_k is 65, i.e., $y_k^M = 65$, then, y_k has to be updated to 65.

Since it is possible that the limitation is too restrictive, in Lines 11 to 14, the algorithm verifies that x_i^M, y_i^M do not violate the guards of the relative PLC and that their span $y_i^M - x_i^M$ contains the contingency c_i. If any guard of a process P_i is violated, then P_i cannot be considered, and the algorithm has to stop.

Line 14 adjusts the lower guard for PLC associated with processes with no upper bound to their durations, i.e., processes having PLC with format $Z_i \xRightarrow{[[x_i, \infty][y'_i, \infty]]\ 0} E_i$. In the parent process, any P_i has a limited duration because the DC checking algorithm imposes it. Therefore, the lower guard of any P_i must be smaller than or equal to the determined y_i^M.

In the second phase, the algorithm determines the possible guarded ranges for each child process $P_i \in P^+$ and stores them in the ordered set R_i (Lines 15 to 20). In detail, for each subprocess $P_i \in P^+$, it may adjust the outer bounds of the ith PLC with the determined values x_i^M, y_i^M and, then, it determines all the allowed guarded ranges for the considered process by an internal **for** cycle.

In the third phase, the algorithm finds which combinations of guarded ranges for all child processes in P^+ guarantee the dynamic controllability of the parent process. In detail, once all R_i are determined, the algorithm builds the Cartesian product R^* of all possible guarded ranges in line 21. Each row r of S^* has the form
$\langle \big[[x_1^M, x'_1][y'_1, y_1^M]\big]^r, \big[[x_2^M, x'_2][y'_2, y_2^M]\big]^r, \ldots, \big[[x_k^M, x'_k][y'_k, y_k^M]\big]^r \rangle$,
where $\big[[x_i^M, x'_i][y'_i, y_i^M]\big]^r$ is the rth element in the ordered set $S_i, i = 1, \ldots, k$. In other words, each row r of R^* is a complete specification of the guarded ranges for all child processes in the set P^+.

Then, in Lines 22 to 30, it checks the DC property of the parent process using each complete specification of guarded ranges in S^* to accumulate n successful checks. Given the rth row in r^*, the algorithm replace each requirement link $A_i \xrightarrow{[x_i^M, y_i^M]} C_i$ in S_M with the guarded link $A_i \xRightarrow{[[x_i^M, x'_i][y'_i, y_i^M]]^r} C_i$ using the ith element of the rth row in R^*. In this way, the obtained FTNU represents the parent process having all child processes in $P^1 \cup P^+$ configured from the repository. If DC checking determines that the obtained network is DC and no bounds x_i^M, y_i^M in S_M violate the guards of $P_i \in P^0$, then the configuration of the child process is good and added to the set of solutions *SOL*. The child process configuration is discarded if the DC checking returns a negative answer or at least for one i, x_i^M or y_i^M in S_M violates the guards of $P_i \in P^0$. There are three possible outcomes of the third phase. First, there is no good configuration; the algorithm has to check all the possible configurations before returning a negative message. This represents a worst-case running. Second, there are fewer than n good configurations; the algorithm checks all possible configurations and returns the good ones. It is another worst-case running. Third, there are n good configurations, at least; the algorithm checks and returns the first n that represent the configurations with the smallest execution times.

As regards computational complexity, the algorithm implements a brute-force technique for finding good child-process configurations. Therefore, the time complexity is $O(|R^*|M|\mathcal{T}|^4 3^{|\mathcal{P}|}2^{|\mathcal{G}|})$, where $O(M|\mathcal{T}|^4 3^{|\mathcal{P}|}2^{|\mathcal{G}|})$ is the time complexity of a single DC checking (see [14]), and M is the maximum absolute value of any weight in the network.

6.2. Managing DC-violations

Let us now focus on DC violations induced by some child process. Two possible situations are determining DC violations.

- The first one is related to the fact that child processes may have an overall duration exceeding the maximal duration allowed by the parent process.
- A second case is when the duration of a child process is less than the minimum duration required by the parent process.

While in the first case, we cannot use all the selected child processes within the parent process, in the second case, we may have some real-world domain that allows the parent process to "wait" for the end of a (too fast) child process. This waiting behavior could be implemented by introducing a time lag $\mathsf{E}[0, +\infty]\mathsf{S}$ between the child process component and the first subsequent in the parent process.

Example 13. Let us consider the process fragment depicted in Fig. 6(a). It is an excerpt of the process discussed in Section 3, where real-world guarded ranges are considered for child processes $P_{3\&4}$ and P_5, and the shortest guarded range is considered for child process P_6, having as derived prototypal link range $[\![[2,8][10,20]]\!] \updownarrow 3$. It also has a further time lag $\mathsf{E}[9, 15]\mathsf{S}$ between the end of P_6 and the start of T_{10}, to simulate an observed constant delay in the process. Consider now the sum of the guaranteed maximum contingent durations of child processes, $y'_{3\&4} = 20, y'_5 = 20, y'_6 = 11$, the minimum duration 1 of the $\times_1$, and the minimum delay of 9 time units between P_6 and T_{10}. Such a time span is possible because the upper guard values of child processes are always possible. The value of such a time span is 61 and violates the upper bound 60 of the time lag between $P_{3\&4}$ and T_10. In this case, if we need to use all the specified child processes, it would be necessary to modify the overall structure/constraints of the parent process.

On the other hand, Fig. 6(b) depicts a case in which it is impossible to satisfy a minimum duration requirement. The purpose of this excerpt is to ensure that the hemorrhagic test check must be done within one time-unit after the end of P_5 (see the time lag constraint between P_5 and $\times_1$) and, when the test is negative, the process P_6 has to be started within one time-unit (see time lag constraint between $\times_1$ and P_6). Moreover, a guideline requires that the time span between the end of P_5 and the end of P_6 has to be in the range $[28, 35]$.

Since P_6 can last 20 at maximum, $\times_1$ can last three time-units at maximum, and the delays between P_5 and $\times_1$, $\times_1$ and P_6 are one time-units each at maximum, the minimal span 28 between the end of P_5 and the end of P_6 cannot be satisfied and, therefore, the parent process is not DC. In this case, we can guarantee the DC of the parent process if we allow the parent process to wait until after the end of a child process. In general, the wait "activity" is implemented by

1. adding a new dummy task *DT* having 0 duration[3] after the considered child process,
2. connecting the child process to *DT* by a new sequence flow (edge) (this edge realizes a customizable delay $[0, \infty]$),
3. moving the existing outgoing sequence flow edge from the child process to *DT*, and
4. moving all possible time lag constraints incident to the end of the child process to *DT*, as depicted in Fig. 6(c).

In some schemata, it is possible to simulate a delay exploiting the already present sequence flow edge after the child process. For example, in Fig. 6(b), it is sufficient to change the time lag $P_5 \xrightarrow{E[28,35]E} P_6$ to $P_5 \xrightarrow{E[28,35]S} \times_2$ for having the necessary delay without significantly modifying the parent process.

6.3. Making child processes even more flexible

In the previous section (Section 6.2), we show that allowing a $[0, \infty]$ delay after a child process is possible to satisfy minimum duration constraints involving it when its PLC is not sufficient.

If such a delay is added inside the child process before its end event, we obtain the same result, but its PLC changes its values and characteristics. Indeed, it loses the contingency value and allows single-value contingent cores.

To add a final delay inside a child process, we can similarly operate as explained in Section 6.2 for the parent process:

1. a dummy task *DT* replaces the end event *E*,
2. a new end event *E* is created, and
3. a sequence flow edge connecting *DT* to *E* is added (it represents a delay $[0, \infty]$) to complete the schema of the child process.

Let us now consider the original PLC $Z \xrightarrow{[\![[x,x'][y',y]]\!]\updownarrow c} E$ representing the temporal behavior of the considered child process. After adding delay $[0, \infty]$, the original upper bound y becomes $y = \infty$ as well as the lower guard $x' = \infty$ because the lower guard represents the maximum duration of the process when all possible guarded links last their lower guard value. Moreover, the presence of the final delay allows the compensation of a possible contingency of the processes (see Section 5.3). Therefore, the PLC of the augmented child process always becomes a PLC $Z \xrightarrow{[\![[x,\infty][y',\infty]]\!]\updownarrow 0} E'$, where $x' = y = \infty$ and $c = 0$.

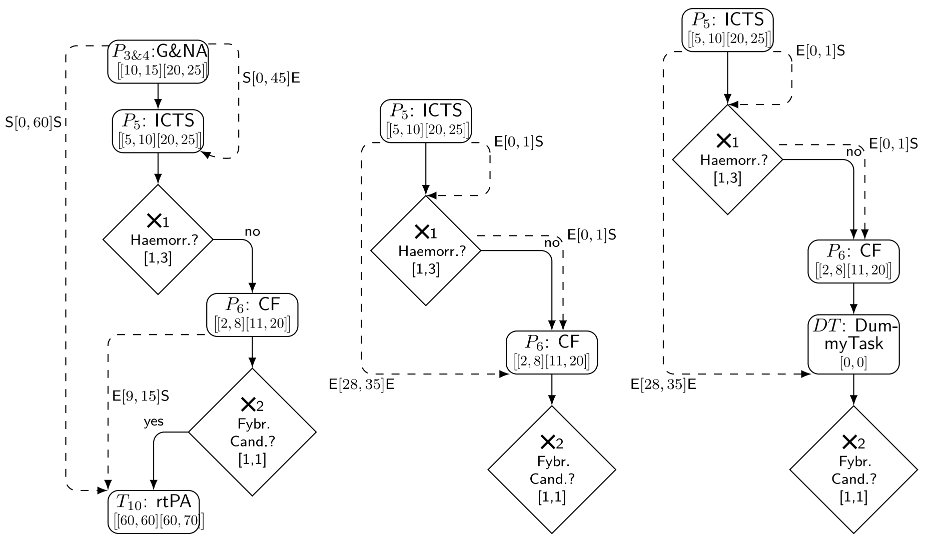
Example 14. Process P_6 (see Fig. 1(c)) can be represented as a prototypal link with contingency having range $[\![[2,8][10,20]]\!] \updownarrow 3$.

Fig. 7 depicts P_6 augmented by an $[0, \infty]$ delay after its original end event that becomes a generic event.

The PLC of the augmented P_6 becomes a PLC where $x' = y = \infty$ and contingency is 0: $Z \xrightarrow{[\![[2,\infty][10,\infty]]\!]\updownarrow 0} E$.

In general, given a PLC $Z \xrightarrow{[\![[x,x'][y',y]]\!]\updownarrow c} E$, it is necessary to select two values x^*, y^* such that $x \le x^* \le x', y' \le y^* \le y$, and $y^* - x^* \ge c$ to have an appropriate guarded range $[\![[x,x^*][y^*,y]]\!]$

[3] Such dummy task represents a zero-duration requirement constraint. Even though it is not correct according to the formal model we introduced, it is used to simplify the notation.



(a) Excerpt of the parent process having real-world guarded ranges for child processes $P_{3\&4}$ and P_5 and a further time lag between P_6 and T_{10}. The guarded range assigned to child processes P_6 is the shortest one among those allowed according to the derived prototypal link.

(b) Excerpt of the parent process with P_6 and the closest gateways. Time lags have been added to simulate scenarios where P_6 is merged within a parent process, requiring wider time spans, concerning the ones represented by the derived prototypal link.

(c) The process excerpt represented in Figure 6b, extended with a dummy task to manage a kind of waiting time inside the parent process, which has to call too fast child processes.

Fig. 6. Two examples of DC-violations when considering child processes.

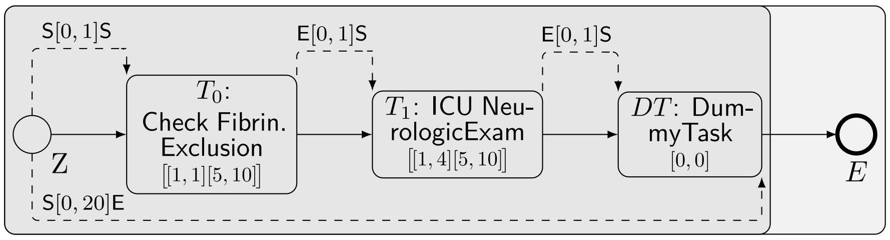


Fig. 7. Child process P_6 augmented by an unlimited delay.

that can be used to represent the behavior of the child process in the parent one.

A temporal constraint given by the contingent range $[x^*, y^*]$ is satisfiable when all other time constraints are satisfiable for each value in $[x^*, y^*]$. Consequently, the smaller the range $[x^*, y^*]$, the easier it is to determine solutions for the network.

The singular aspect of a PLC like $Z\xrightarrow{[[x,\infty][y',\infty]]\updownarrow 0}E$ is that it always admits contingent range of a single value like $\big[[x, b][b, \infty]\big]$, where $y' \le b < \infty$ (In fact, as we have in Algorithm 2, embedding a child process into a parent always forces y to be a finite value, the horizon of the parent process in the worst case).

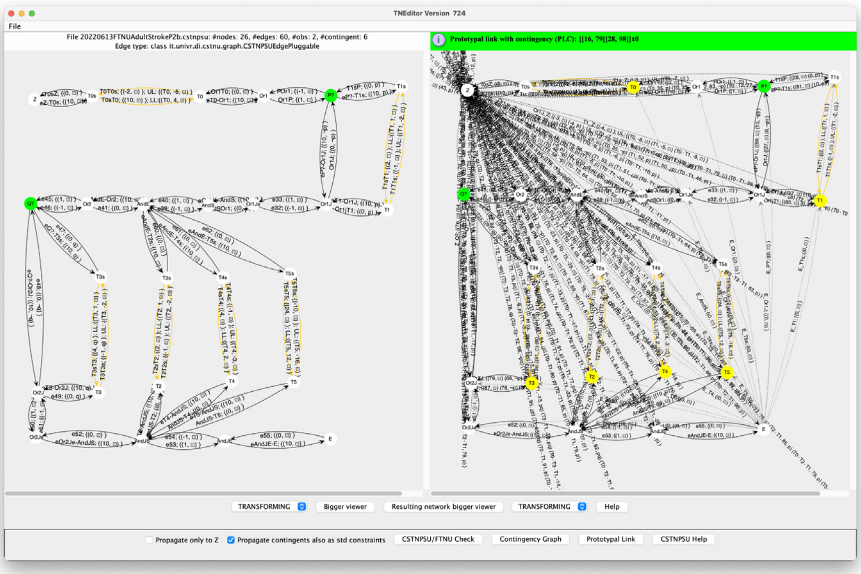
In this way, it is possible to insert the child process into the parent, requiring all constraints in the parent process to be satisfied with respect to the value b only.

This solution is appropriate only when it is necessary to guarantee a minimal duration b that the original child process cannot provide. Otherwise, the downside of this configuration is that the child process is always assumed to last b time units, even though all internal activities end before.

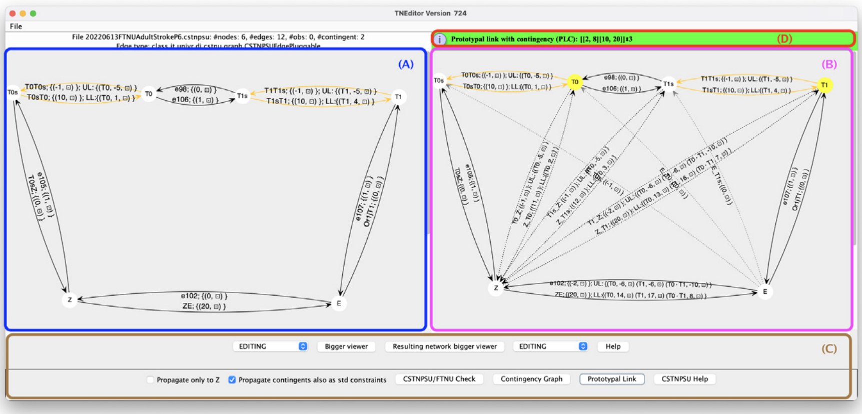
7. Proof-of-concept prototype

As a proof-of-concept, we implemented the algorithm for determining the PLC of a process. In particular, such an algorithm determines the contingency graph associated with the complete FTNU of the process, then the labeled distances of each node from Z in the contingency graph (see Algorithm 1), and, then, the PLC using the values on the complete FTNU and the path contingency span of the last node. This algorithm is a core function to determine the overall behavior of a process in a more high-level application.

The implementation was realized by expanding the CSTNU-Tool [28]. CSTNU-Tool is a Java library (with a simple GUI application to use the main functions) that implements many algorithms related to Simple Temporal Networks (STN) [29], Simple Temporal Networks with Uncertainty (STNU) [30–32], Conditional Simple Temporal Problem (CSTN) [20,33,34], Conditional Simple Temporal Problem with Uncertainty (CSTNU) [15,16,35], Flexible Temporal Network with Uncertainty (FTNU) [14], and others.


TNEditor Version 724
File
File 20220613FTNUAdultStrokeP2b.cstnpsu: #nodes: 26, #edges: 60, #obs: 2, #contingent: 6
Edge type: class it.univr.di.cstnu.graph.CSTNPSUEdgePluggable
Prototypal link with contingency (PLC): [[16, 79][28, 98]]±0
TRANSFORMING
Bigger viewer
Resulting network bigger viewer
TRANSFORMING
Help
Propagate only to Z
Propagate contingents also as std constraints
CSTNPSU/FTNU Check
Contingency Graph
Prototypal Link
CSTNPSU Help

(a) PLC computation result when input is FTNU P_2 of Figure 2a.


TNEditor Version 724
File
File 20220613FTNUAdultStrokeP6.cstnpsu: #nodes: 6, #edges: 12, #obs: 0, #contingent: 2
Prototypal link with contingency (PLC): [[2, 8][10, 20]]±3
(A)
(B)
(C)
(D)
EDITING
Bigger viewer
Resulting network bigger viewer
EDITING
Help
Propagate only to Z
Propagate contingents also as std constraints
CSTNPSU/FTNU Check
Contingency Graph
Prototypal Link
CSTNPSU Help

(b) PLC computation result when input is FTNU P_6 of Figure 2b.

Fig. 8. CSTNU-Tool screenshots.

Class FTNU.java contains the implementation of some algorithms for checking the DC controllability and determining the minimized version of the FTNU instance. We extended such a class by adding three methods:

- getContingencyGraph for determining the path contingency graph of an FTNU instance;
- getEMaxDistanceInContingencyGraph for determining the labeled path consistency span of the last node of an FTNU instance;
- getPrototypalLink for determining the PLC of an FTNU instance;
- configureSubNetworks for determining all possible temporal configurations of child processes in a parent one (see Algorithm 2).

Then, we extended the GUI application TNEditor to allow users to use such new methods in a simple environment.

Fig. 8 depicts the screenshots of TNEditor after the computation of the prototypal link with the contingency of two different inputs, the FTNU representing the process P_2 (Fig. 2(a)) and the FTNU representing the process P_6 (Fig. 2(b)).

The application window can be divided into three main parts. Let us describe these main parts considering Fig. 8(b). Frame (A) depicts the input FTNU instance loaded or created inside the application, while frame (B) shows the result of any elaboration on the input instance made using the commands and options present in frame (C).

In particular, in frame (C):

- the option Propagate only to Z allows a fast DC checking (when selected) or a (slower) complete DC checking.
- the CSTNPSU/FTNU Check button activates the DC checking; once the DC checking algorithm finishes its computation, the resulting network is shown in frame (B) with a comment in frame (D).
- the Contingency Graph button allows the determination of the contingency graph of a DC instance; it activates a DC checking and, then, the determination of the contingency graph if the instance is DC.
- the Prototypal Link button allows the determination of the PLC of the process. When clicked, it shows the complete FTNU instance in frame (B) and the determined PLC in frame

(D) if the instance is DC; otherwise, it shows the partially checked instance in frame (B) and an error message in frame (D).

The computation time of each procedure for the shown processes is a few milliseconds. In the next section, we provide an experimental evaluation of computation time for instances of greater size.

Version 4.9 of the CSTNU-Tool [28] containing the implementation of the above algorithm is freely available at https://profs.scienze.univr.it/~posenato/software/cstnu.

8. Empirical performance evaluation

This section presents an empirical evaluation of the performance of the FTNU DC-checking algorithm and of the `getPrototypalLink` procedure presented in Section 7.

We recall that the `getPrototypalLink` procedure, given a DC instance as input, has to determine the completion of the instance (Definition 4) and the path contingency span of each node (Algorithm 1) to calculate the prototypal link.

The comparison of the two algorithm performances should give an idea of the computational cost for having a compact representation of a (sub)process versus the cost of determining its controllability only.

The tests were executed using a Java Virtual Machine 17 on an Apple PowerBook (M1 Pro processor) configured to use 8 GB memory as heap space. The source code and benchmarks are freely available [28].

For testing, we considered and expanded the benchmarks presented in [16] for evaluating the performance of CSTNU DC-checking algorithms. In particular, in [16] the authors proposed three benchmarks (B3, B4, and B5), each having 250 DC CSTNUs and 250 non-DC CSTNUs. Such CSTNU instances were obtained from random workflow schemata generated by the ATAPIS toolset [36]. Benchmark B3 contains instances derived from random workflows, having $N = 10$ activities (represented as contingent links) and $k = 3$ XOR gateways (represented as observation timepoints). Benchmark B4 and B5 are similar to B3, but B4 instances contain 4 XOR gateways each, while B5 instances contain 5 XOR gateways each. Each benchmark, B3, B4, and B5, is subdivided into five sub-benchmarks, Bj_i with $j = 3, 4, 5$ and $i = 0, 1, 2, 3, 4$, each one having 50 instances, generated by also fixing the number of AND gateways to value 0, 1, 2, 3, and 4, respectively to evaluate the impact of the parallel components (AND gateways). It is shown that given a workflow instance having N tasks, k XOR connectors, and j AND gateways, the corresponding CSTNU instance has $5 + 2N + 6k + 6j$ nodes, N contingent links, and k observations.

Starting from B3, B4, and B5, we built a new set of 3 benchmarks (limited to DC instances) where each CSTNU instance was transformed into an FTNU one converting each contingent link to a guarded one. The conversion consisted of adding two outer bounds (lower and upper) to each contingent link. In particular, each contingent link (A, x, y, C) was replaced by the guarded link $(A, \big[[x'', x][y, y'']\big], C)$, where x'' was set to $(1 - r)x$ while y'' was set to $(1 + r)y$, for a suitable r. In this way, it is guaranteed that every CSTNU controllable instance is converted into an FTNU controllable one. We tested different values of r, and we noted no significant changes in the execution time of the algorithms. Therefore, we fixed r to 10% of the given duration of the contingent link, to build the final benchmarks.

Fig. 9 displays the average execution times of the two algorithms over all five sub-benchmarks in B3, B4, and B5.

Each data point value is the sample average $\bar{X}_{50} = \frac{\sum_{i=1}^{50} X_i}{50}$ of average execution times X_i obtained considering the fifty instances of the relative sub benchmark. Indeed, each X_i is the average execution time obtained executing five times the algorithm on instance having index i in the considered sub benchmark. The error bar represents a 95% confidence interval for the average execution time of the algorithm on instances of the considered sub-benchmark.

As concerns the FTNU DC checking performance, from the data in Fig. 9, it results that the performance is similar to the one obtained for the CSTNU DC checking algorithm in [16] although here the average times are one-order of magnitude smaller (thanks to the M1 processor). The more difficult instances are associated with workflows without parallel gateways (i.e., instances in the first sub-benchmark of each main benchmark) and the algorithm performs better as the number of AND gateways increases, but in B5. As stated in [37], such behavior is due to how the ATAPIS random generator works when the number of AND gateways is small (i.e., less than 5). Increasing the number of AND gateways (till 5), fewer XOR gateways are set in sequence and, therefore, there are fewer possible scenarios. In $B5_4$, where the number of AND gateways is 4, this pattern did not occur. The sub-benchmark contains many instances with three-four observation timepoints over five in sequence, determining a greater number of possible scenarios and, hence, a greater execution time for the checking.

As concerns the `getPrototypalLink` procedure, its execution times are much lower than those of the DC checking algorithm (see Fig. 9). Fig. 10 shows the average execution time of `getPrototypalLink` of Fig. 9 in linear y-scale. Once a network is checked DC, the completion phase updates the values of the original guarded and requirements links in the network while the building of the path contingency span graph creates and fills a vector of labeled distances from Z to each node. Such phases require visiting each original edge of the network two times, each time considering all the labeled values associated with the edge. We verified that the average node degree is less than 5 in all benchmarks, hence the instances are sparse graphs. In these benchmarks, the quantity of labeled values present in each edge is not relevant as the number of edges/nodes in the determination of the computation time. Therefore, the `getPrototypalLink`-performance results to be quasi-linear with respect to the number of nodes.

8.1. Composition of two FTNUs

In this section, we want to show a practical example of how our methodology can help to study networks composed of sub-networks even when a composition is simple.

Let us consider two of the most difficult (from the point of DC determination time) instances of B3: `template_006` and `template_021`. Table 5 shows the main structural and temporal characteristics of the two instances: n is the number of nodes, m is the number of edges, g is the number of guarded links, k is the number of propositions, and Time is the execution time for DC-checking and prototypal link determination.

Now, let us suppose to want to verify if it is possible to compose the two networks in a sequential way such that the duration of the whole process is within a given bound. More precisely, we want to verify the DC controllability of a network, called `template_006_021`, where there are:

- all nodes and edges of `template_006`,
- all nodes and edges of `template_021` (renamed properly),
- a constraint that forces the starting node of `template_021` to be a one-time unit after the ending node of `template_006`,
- a constraint that limits the global duration of the network to 500 time-units.

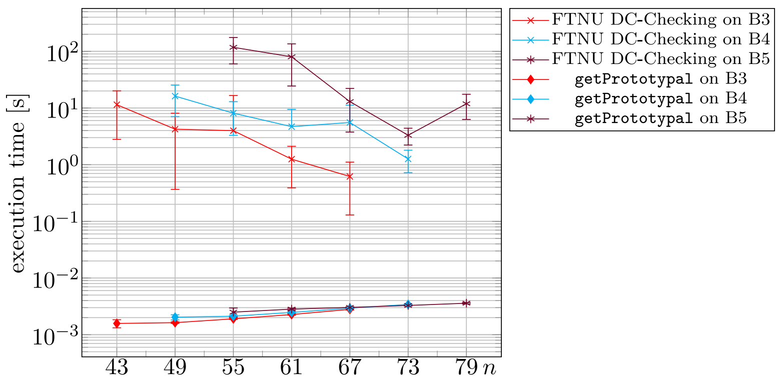


Fig. 9. Benchmark B3, B4, and B5.

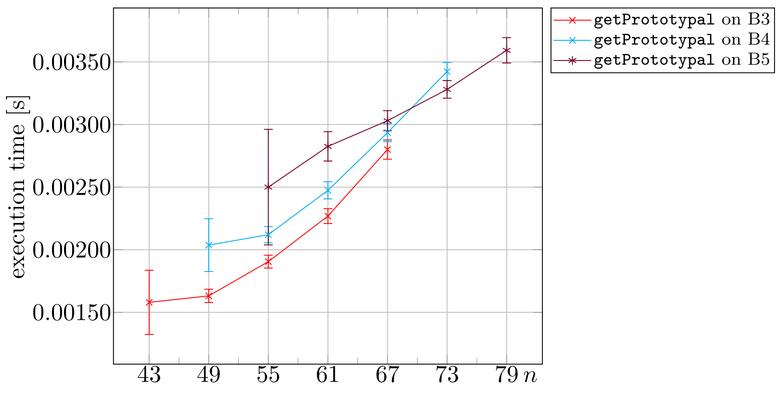


Fig. 10. `getPrototypalLink` average execution time detail.

Table 5
Structural and Temporal Characteristics of instances `template_006` and `template_021`.

Name	n	m	g	k	Prototypal link	Time [s]
template_006	43	126	10	3	$[\![119, 818][230, 1070]\!]\updownarrow 0$	61.66 ± 0.68
template_021	43	130	10	3	$[\![92, 856][219, 1033]\!]\updownarrow 0$	42.13 ± 1.63

Network `template_006_021` has 88 nodes (43*2 + starting and ending nodes), 260 edges, 20 guarded links, and 6 propositions. It results that checking the DC controllability of `template_006_021` requires more than 1200 s (time out) making impossible the evaluation of its controllability in a reasonable time.

Now, let us consider another network, `template_parent_006_021`, where there is a sequence of two guarded links representing the two sub-networks (see Fig. 11). The values for the two guarded links in Fig. 11 have been determined as the first solution (combination) by Algorithm 2.

The execution of Algorithm 2 (limited to determine only one solution) required $\approx$ 0.00065 s. The resulting prototypal link of the parent process resulted to be $[\![214, 500][452, 500]\!] \updownarrow 0$, while the guarded link representing `template_006` was updated to $[\![119, 230][230, 278]\!]$ and the guarded link representing `template_021` was updated $[\![92, 219][219, 267]\!]$.

Since the guarded links determined for the two sub-networks are more restrictive than the relative prototypal links, is necessary to propagate such restrictions to the sub-networks before the execution for having the two sub-networks ready.

The propagation of each of such determined guarded links is obtained by

- setting a constraint link between the starting node and the end one of the sub-network with values equal to the outer values of the guarded link, and
- propagating such a new constraint by executing a DC checking of the sub-network.

Such updates can be done in less than 120 s (see the DC checking times in the previous table).

Therefore, in less than 120 s it is possible to check the DC controllability and to configure the sub-networks if they are managed in a composite way, while it is necessary more than 1200 s to check the network if it contains all nodes and edges of the two subnetworks.

In summary, we verified experimentally that the computational cost of the prototypal link of an FTNU is negligible with respect to the DC-checking computational one. Therefore, subdividing a complex process into smaller subprocesses (requiring smaller DC checking costs) and checking the overall controllability using our proposed approach can help limit the overall computational cost of determining the overall dynamic controllability.

9. Related work

9.1. Time and business processes

The management and the representation of temporal features of business processes have been considered for some years from different perspectives: from conceptual modeling to checking

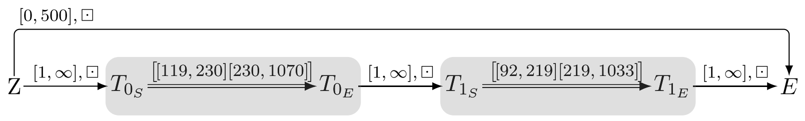


Fig. 11. `template_parent_006_021`.

temporal consistency and/or controllability, to temporal agreements among different organizations, to run-time verification of temporal properties, to temporal constraints and dynamic changes in business process schemata [6,8,10–12,25,38,39]. To summarize all the different approaches, we may say that most of them adopt some expressive (diagrammatic) models/languages to represent tasks, subprocesses, routing behaviors, and control flows, suitably extended also to consider temporal properties. BPMN is widely adopted in these contexts. Then, suitable mappings of such expressive models are proposed for lower-level models, where different temporal properties can be checked through suitable algorithms. Different kinds of temporal constraint networks, Petri nets, queuing models, etc., are used for this purpose.

Regarding the checked temporal properties, *temporal controllability* was originally studied for temporal constraint networks. In [4], the authors proposed dynamic controllability of time-aware process schemata. In [8], in turn, the authors extended their work, analyzing the computational complexity of the *dynamic controllability problem*, proposing a general algorithm for checking the dynamic controllability of a time-aware process schema.

No one of these proposals allows the representation of flexible contingent durations for tasks and moreover, no one focuses on the issue of representing in a compact way the overall temporal behavior of a (child) process. Thus, DC-checking of such temporally flexible business (child) processes is beyond the scope of these proposals. It is also worth noting that, at the lower level of network-based representation, DC-checking of temporal constraint networks with guarded links has been introduced quite recently [14] and no other approaches were at the disposal to analyze such kinds of networks.

9.2. Process modularity

Process modularity is an important issue because of its consequences in the reuse, sharing, management, and maintainability of business process schemata. Different aspects and approaches are presented in the literature, e.g., focusing on identifying reusable tasks [40]; applying Aspect-Oriented concepts to design modular business processes [41]; discovering complex data-intensive BPMN subprocesses [42]; identifying parts and patterns of use of business processes and building a catalog of reusable components [40].

Despite such different research directions toward modularity for business processes, the problem of modularity concerning temporal behaviors of processes has not yet received the attention it deserves. Among the past contributions dealing with this issue, we mention here the work of Lanz and colleagues. Although they did not directly face the problem of temporal features of subprocesses, they dealt with the specification of common time patterns in time-aware processes. According to a software-engineering perspective, it can be considered a complementary research direction toward reusing well-known solutions in process design.

More specifically, Lanz et al. [7][43] discussed 10 time patterns. Such patterns represent common temporal constraints of time-aware processes and are provided with formal semantics, allowing their reuse without ambiguities. The need for sophisticated run-time support for the time patterns is discussed in [7].

Moving to the issue of modularity and time-aware subprocesses, a "dual" approach concerning the one discussed in this paper is proposed in [44], where the authors use the modularity capabilities of BPMN to manage temporal constraints for single tasks or overall single-entry single-exit (SESE) process regions. Subprocesses are used to "envelope" specific tasks (or SESE regions) with time-based events and signal events, which allow the management of temporal constraints, expressed as minimum and maximum time distance between two different (starting and ending) timepoints. This way, complex temporal constraints are expressed in BPMN and, at the same time, are hidden from the external stakeholders, who are interested in application-oriented features like deadlines and so on, and not in how they are represented and checked in the process model.

Finally, in [45], the concept of "action refinement" is proposed, which replaces atomic actions with subprocesses. Here, the authors extend timed Dynamic Condition Response (DCR) graphs to contain subprocesses resulting from stepwise refinement. They specify when substituting an event (a task, in our terminology) with a subprocess is feasible without any issue in the resulting overall process. As DCR graphs can represent deadlines and temporal constraints, we may consider such a proposal the closest one to the proposal we make in this paper. However, different modeling choices make this approach only partially comparable with our proposal. Indeed, in DCR graphs, events are atomic concepts that occur in a time unit. Thus, contingent durations are not representable in that context. Moreover, structured process models cannot be specified as gateways are not considered. On the other hand, exclusion/inclusion relationships allow for different possible runs of a graph, which are not directly considered in BPMN-based models.

Such contributions towards modularity and temporal issues for business processes do not consider any kind of controllability. Indeed, temporal constraints and (guarded) contingent durations cannot be completely represented and the problem of checking temporal constraints without "unfolding" child processes is not in their specific focus.

10. Discussion and conclusions

In this paper, we proposed a novel approach for determining and representing the overall temporal behavior of a process, called *prototypal link with contingency*. A prototypal link with contingency allows the determination of all possible durations of a (child-)process and any permissible restriction that may be applied to them while still ensuring the executability of the process.

Some specific aspects of our research deserve some discussion, in order to identify limitations and specific research foci that characterize our proposal.

XOR split. Our proposal considers only XOR gateways with a binary condition. Gateways with multiple exclusive conditions require to be modeled through a cascade of binary XOR gateways. On the other hand, inclusive OR gateways have not been considered and should be represented through an AND gateway containing many parallel suitable XOR binary conditions. In general such complex OR behaviors would require further research efforts and suitable mappings to many simpler constructs.

Cycles and isolated tasks. We did not explicitly consider cycles and isolated tasks in our proposal. As for isolated tasks, not having any order of execution with respect to other tasks of a business process, they can be suitably represented and possible time lags with respect to other tasks can be managed. Checking the temporal features of cycles is a challenging feature. While cycles without any predefined limitation on the number of iterations cannot be checked, cycles with a limited number of iterations (or an overall maximum duration) may be accommodated in our proposal by "unfolding" them and explicitly representing all the possible cycle executions connected through nested XOR gateways. Such, possibly complicated, "unfolding" would be done internally, after the specification of the cycle by the user.

Declarative process languages. In this paper, we explicitly focused on the imperative process language BPMN. Declarative process languages, where the order of execution of tasks not given a priori and strongly depends on the knowledge and the decisions of participants [46], would benefit from our proposal, which adds some kind of temporal flexibility. However, the representation of temporal features (task durations and temporal constraints) should be carefully considered, to analyze and represent how such temporalities are intertwined with other kinds of dependencies among tasks.

Main results and future work. To summarize the main contributions of this paper:

1. We showed that prototypal links with contingency can provide a compact representation of the temporal behavior of processes where the set of executed activities can change according to some runtime conditions (and it extends a previous contribution [9], where XOR splits, i.e., multiple execution paths, were not allowed.)
2. We proposed an efficient method for determining prototypal links with contingency based on the FTNU model, a temporal constraint network model that allows the representation of enriched contingent constraints and admits a compact checking and execution algorithm.
3. We proposed an algorithm for determining all possible configurations of prototypal links with the contingency of all (child-)processes that guarantee a successful execution (i.e., dynamic controllability) of the global process.
4. We discussed when and how it is possible to adjust the parent/(child-) processes structure to guarantee a successful execution when there are no successful configurations of prototypal links with contingency.
5. As a proof-of-concept, we provided the algorithm implementation that determines the prototypal link with the contingency of a process with conditions inside the CSTNU Tool library. We performed an extended empirical evaluation of the proposed algorithms, showing their practical applicability.

In future work, we want to study the integration of time-aware processes in PAISs, specifically focusing on aspects like scalability and usability, even in the presence of data-intensive and decision-intensive processes.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request

Appendix. Proofs of Lemmas 1 and 3

In this appendix, we introduce proofs of lemmas we used to prove Theorem 2. Such proofs use some propagation rules for determining if a network is dynamically controllable [14]. For completeness, we report such propagation rules, Table A.6, without explaining them; a complete description of the rules can be found in [14].

A.1. Proof of Lemma 1

Proof. Let us assume that X has a positive path contingency span, $\mathrm{cont}_S(X) > 0$, and $y' - x' < \mathrm{cont}_S(X)$.

By the definition of $\mathrm{cont}_S()$, there is a sequence of timepoints $Y_0, \dots, Y_k$ with $Y_k \equiv X$ such that

$$\mathrm{cont}_S(C) = \mathrm{cont}_S(Y_{0_{\ell_0}}) + \Delta_{Y_0 Y_{1_\boxdot}} + \cdots + \Delta_{Y_{k-1} Y_{k_{\ell_k}}}$$

where

1. $\ell_0 \ell_1 \cdots \ell_k \in \mathcal{P}^*$ is a consistent label,
2. $\mathrm{cont}_S(Y_{0_{\ell_0}}) = 0$,
3. $\forall j \in \{1, \dots, k\} \sum_{i \in \{1,\dots,j\}} \Delta_{Y_{i-1} Y_{k_{\ell_i}}} > 0$.

By item 3 and Δ definition, link $Y_0 Y_1$ is s guarded link $Y_0 \xRightarrow{[[x_1, x'_1][y'_1, y_1]]} Y_1$.

Let us assume that path $Y_0, \dots, Y_k$ is made by alternating guarded and requirement links. Indeed, suppose there are more consecutive requirement links. In that case, we can replace such a sequence with a single requirement link having as range the sum of the ranges and label the conjunction of labels. At the same time, if there are more consecutive guarded links, we can be separated each pair of them by a requirement link having range $[0, 0], \boxdot$; if the path ends with a guarded link, we can add a requirement one with $[0, 0], \boxdot$ range.

Therefore, we can assume that the sequence of timepoints $Y_0, \dots, Y_k$ always has the following pattern:

$$Z \overset{[a,b],\,\ell_0}{\rightsquigarrow} Y_0 \xRightarrow{[[x_1, x'_1][y'_1, y_1]]} Y_1 \xrightarrow{[x_2, y_2],\,\ell_2} Y_2 \xRightarrow{[[x_3, x'_3][y'_3, y_3]]} Y_3 \xrightarrow{[x_4, y_4]\ell_4} Y_4 \dots$$
$$\dots \xRightarrow{[[x_{k-1}, x'_{k-1}][y'_{k-1}, y_{k-1}]]} Y_{k-1} \xrightarrow{[x_k, y_k],\,\ell_k} Y_k \equiv X$$

where $Z \overset{[a,b],\,\ell_0}{\rightsquigarrow} Y_0$ is the requirement link derived by the DC checking algorithm.

It is possible to show, by induction, that it is not possible to restrict $Z \xrightarrow{[u,v],\,\ell} Y_k$ to $[u^*, v^*]$ where $v^* - u^* < \mathrm{cont}_S(X)$ and ℓ is consistent with $\ell_0 \ell_1 \cdots \ell_k$.

Let us assume that $v^* - u^* = \mathrm{cont}_S(X) - \epsilon$, $\epsilon > 0$. Then, we show that at least one link inside path $Z, Y_0, \dots, Y_k$ has to be restricted beyond its bounds/guards.

Base. Path of three timepoints Y_0, Y_1, and Y_2:

$$Z \overset{[a,b],\,\ell_0}{\rightsquigarrow} Y_0 \xRightarrow{[[x_1, x'_1][y'_1, y_1]]} Y_1 \xrightarrow{[x_2, y_2],\,\ell_2} Y_2$$

In this case, $\mathrm{cont}_S(Y_{2_{\ell^*}}) = \mathrm{cont}_S(Y_{0_{l_0}}) + (y'_1 - x'_1) + (x_2 - y_2)$, where $\mathrm{cont}_S(Y_{0_{\ell_0}}) = 0$ and $\ell^* = \ell_0 \ell_2$. Assume $Z \xrightarrow{[u,v],\,\ell'} Y_2$ is restricted to $Z \xrightarrow{[u^*,v^*],\,\ell'} Y_2$ where $v^* - u^* = \mathrm{cont}_S(Y_{2^*_\ell}) - \epsilon = (y'_1 - x'_1) + (x_2 - y_2) - \epsilon$, $\epsilon > 0$ and ℓ' is consistent with ℓ^*.

Let us consider the corresponding distance graph where we determine some new constraints from the given ones.

Rule rG_1 and rG_5 (cf. Table A.6) allow the determination of ordinary edges $Z \xrightarrow{\langle v^* - x_2,\, \ell'\ell_2 \rangle} Y_1$ and $Z \xleftarrow{\langle -(u^* - y_2),\, \ell'\ell_2 \rangle} Y_1$ between Z and Y_1.

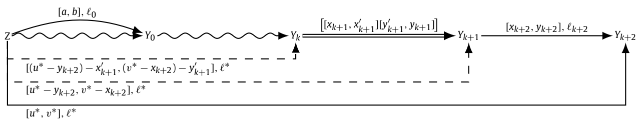
Rule rG_2 derives an ordinary edge $Z \xleftarrow{\langle x'_1 - (u^* - y_2),\, \ell'\ell_2 \rangle} Y_0$.

Rule rG_7 derives an ordinary edge $Z \xrightarrow{\langle (v^* - x_2) - y'_1,\, \ell'\ell_2 \rangle} Y_0$.

As the network is DC, there can be no negative circuit between Z and Y_0, i.e., $(v^* - x_2 - y'_1) + (x'_1 - u^* + y_2) \geq 0$.

Now, $(v^* - x_2 - y'_1) + (x'_1 - u^* + y_2) \geq 0$ implies $v^* - u^* \geq (y'_1 - x'_1) + (x_2 - y_2)$, that contradicts the hypothesis that $v^* - u^* = (y'_1 - x'_1) + (x_2 - y_2) - \epsilon$. The network can no longer be DC as the requirement link $Z \xrightarrow{[u,\,v],\,\ell'} Y_2$ is restricted too much.

Inductive step. Let us consider a path consisting of $k + 3$ time-points $Y_0, e_0, Y_1, e_1, \ldots, Y_{k+2}$ as depicted below


[a, b], ℓ0
Z
Y0
Yk
[[xk+1, x'k+1][y'k+1, yk+1]]
Yk+1
[xk+2, yk+2], ℓk+2
Yk+2
[(u* − yk+2) − x'k+1, (v* − xk+2) − y'k+1], ℓ*
[u* − yk+2, v* − xk+2], ℓ*
[u*, v*], ℓ*

where $\ell^* = \ell_0\ell_1 \cdots \ell_{k+2}$ is the consistent label that characterizes the path.

Let us assume the path $Z, Y_0, \ldots, Y_{k+2}$ is the path that has the maximum path contingency path, i.e., $\mathrm{cont}_S(Y_{k+2_{\ell^*}}) = \mathrm{cont}_S(Y_{k+2})$. By construction, each node in such a path has the path contingency path equal to its maximum. Then, let us assume that the $Z \xrightarrow{[u,\,v],\,\ell^*} Y_{k+2}$ is restricted to $Z \xrightarrow{[u^*,\,v^*],\,\ell^*} Y_{k+2}$ with $v^* - u^* = \mathrm{cont}_S(Y_{k+2}) - \epsilon$, where $\epsilon > 0$. Note that it is possible to consider any propositional label entailed by ℓ^* instead of ℓ^*. Without loss of generality, we consider ℓ^* to simplify the notation.

Considering the representation of the subpath $Y_{k+1}Y_{k+2}Z$ in the associated distance graph, it is possible to apply rule rG_1 and rG_5 (cf. Table A.6) to derive the two ordinary edges $Z \xrightarrow{\langle v^* - x_{k+2},\, \ell^*\rangle} Y_{k+1}$ and $Z \xleftarrow{\langle -(u^* - y_{k+2}),\, \ell^*\rangle} Y_{k+1}$ between Z and Y_{k+1}.

Rule rG_2 derives the ordinary edge $Z \xleftarrow{\langle x'_{k+1} - (u^* - y_{k+2}),\, \ell^*\rangle} Y_k$, while rule rG_7 derives the ordinary edge $Z \xrightarrow{\langle (v^* - x_{k+2}) - y'_{k+1},\, \ell^*\rangle} Y_k$.

Such new edge constraints determine in the FTNU graph a requirement link $Z \xrightarrow{[(u^* - y_{k+2}) - x'_{k+1},\, (v^* - x_{k+2}) - y'_{k+1}]} Y_k$.

Thus the requirement link $Z \xrightarrow{[a,\,b],\,\ell^*} Y_k$ between Z and Y_k derived by the DC-checking algorithm has a span:

$$\begin{aligned}
b - a &\leq (v^* - x_{k+2}) - y'_{k+1} - ((u^* - y_{k+2}) - x'_{k+1})\\
&= (v^* - u^*) - (y'_{k+1} - x'_{k+1}) - (x_{k+2} - y_{k+2})\\
&= \mathrm{cont}_S(Y_{k+2}) - \epsilon - \Delta_{Y_kY_{k+1_{\ell^*}}} - \Delta_{Y_{k+1}Y_{k+2_{\ell^*}}}\\
&\quad // v^* - u^* = \mathrm{cont}_S(Y_{k+2}) - \epsilon \text{ by hypothesis}\\
&= \mathrm{cont}_S(Y_k) + \Delta_{Y_kY_{k+1_{\ell^*}}} + \Delta_{Y_{k+1}Y_{k+2_{\ell^*}}}\\
&\quad - \epsilon - \Delta_{Y_kY_{k+1_{\ell^*}}} - \Delta_{Y_{k+1}Y_{k+2_{\ell^*}}}\\
&\quad // \text{by Definition 6}\\
&= \mathrm{cont}_S(Y_k) - \epsilon
\end{aligned}$$

Thus, the range is restricted such that $b - a \leq \mathrm{cont}_S(Y_k) - \epsilon < \mathrm{cont}_S(Y_k)$ holds. By induction, this last result implies that the network is not DC. □

Table A.6
Edge generation rules of the FTNU-DC-Check algorithm.

Rule	Conditions	Pre-existing and Generated Edges
rG_1	$u - v < 0$, $\alpha\beta \in \mathcal{P}^*$	$Z \xleftarrow{\langle \aleph:-v,\, \beta\rangle} Y \xleftarrow{\langle u,\, \alpha\rangle} X$; $Z \xleftarrow[\colorbox{lightgray}{$\langle \aleph:u - v,\, \alpha\beta\rangle$}]{} X$
rG_2	$x - v < 0$, $C \notin \aleph$, $\beta \in \mathcal{P}^*$	$Z \xleftarrow{\langle \aleph:-v,\, \beta\rangle} C \xleftarrow{\langle c{:}x,\, \boxdot\rangle} A$; $Z \xleftarrow[\colorbox{lightgray}{$\langle \aleph:x - v,\, \beta\rangle$}]{} A$
rG_3	$\beta \in \mathcal{P}^*$	$Z \xleftarrow{\langle \aleph:-v,\, \beta\rangle} A \xleftarrow{\langle C:-y',\, \boxdot\rangle} C$; $Z \xleftarrow[\colorbox{lightgray}{$\langle C\aleph:-y' - v,\, \beta\rangle$}]{} C$
rG_4^*	$m = \max\{-v, -w - x\}$, $C \notin \aleph\aleph_1$, and $\beta, \gamma \in \mathcal{Q}^*$.	$Y \xrightarrow[\colorbox{lightgray}{$\langle \aleph\aleph_1:m,\, \beta\star\gamma\rangle$}]{\langle C\aleph:-v,\, \beta\rangle} Z \xleftarrow{\langle \aleph_1:-w,\, \gamma\rangle} A$; $A \xrightarrow{\langle y,\, \boxdot\rangle;\ \langle c{:}x',\, \boxdot\rangle} C$; $A \xleftarrow[\langle -x,\, \boxdot\rangle;\ \langle C:-y',\, \boxdot\rangle]{} C$
rG_5	$u + v \geq 0, \alpha\beta \in \mathcal{P}^*$	$Z \xrightarrow{\langle \daleth{:}v,\, \beta\rangle} Y \xrightarrow{\langle u,\, \alpha\rangle} X$; $Z \xrightarrow[\colorbox{lightgray}{$\langle \daleth{:}v + u,\, \alpha\beta\rangle$}]{} X$
rG_6	$c \notin \daleth$	$Z \xrightarrow{\langle \daleth{:}v,\, \beta\rangle} A \xrightarrow{\langle c{:}x',\, \boxdot\rangle} C$; $Z \xrightarrow[\colorbox{lightgray}{$\langle c\daleth{:}x' + v,\, \beta\rangle$}]{} C$
rG_7	$v - y' \geq 0$, $c \notin \daleth$	$Z \xrightarrow{\langle \daleth{:}v,\, \beta\rangle} C \xrightarrow{\langle C:-y',\, \boxdot\rangle} A$; $Z \xrightarrow[\colorbox{lightgray}{$\langle \daleth{:}v - y',\, \beta\rangle$}]{} A$
rG_8	$C \notin \aleph, c \notin \daleth, \aleph$ and $\daleth$ have no common names, $[(\alpha\beta \in \mathcal{P}^*)$ or $(\alpha \in \mathcal{Q}^\star \setminus \mathcal{P}^\star,\ \beta \in \mathcal{P}^\star$ and α contains $\beta.)]$.	$Z \xleftarrow{\langle \aleph:-v',\, \alpha\rangle} A$; $A \xrightarrow{\colorbox{lightgray}{$\langle v - v',\, \boxdot\rangle$};\ \langle c{:}x',\, \boxdot\rangle} C$; $A \xleftarrow[\langle -x,\, \boxdot\rangle;\ \langle C:-y',\, \boxdot\rangle]{} C$; $Z \xrightarrow[\langle \daleth{:}v,\, \beta\rangle]{} C$
rG_9	$C \notin \aleph, c \notin \daleth, \aleph$ and $\daleth$ have no common names, $[(\alpha\beta \in \mathcal{P}^*)$ or $(\alpha \in \mathcal{Q}^\star \setminus \mathcal{P}^\star,\ \beta \in \mathcal{P}^\star$ and α contains $\beta.)]$.	$Z \xrightarrow{\langle \daleth{:}v,\, \beta\rangle} A$; $A \xrightarrow{\langle y,\, \boxdot\rangle;\ \langle c{:}x',\, \boxdot\rangle} C$; $A \xleftarrow[\colorbox{lightgray}{$\langle v - v',\, \boxdot\rangle$};\ \langle C:-y',\, \boxdot\rangle]{} C$; $Z \xleftarrow[\langle \aleph:-v',\, \alpha\rangle]{} C$
rM_1	$\beta \in \mathcal{Q}^*$, $\tilde{p} \in \{p, \neg p, ?p\}$	$Z \xleftarrow[\colorbox{lightgray}{$\langle \aleph:-w,\, \beta\rangle$}]{\langle \aleph:-w,\, \beta\tilde{p}\rangle} P?$
rM_2	$m = \max\{-v, -w\}$ $\beta, \gamma \in \mathcal{Q}^*$, $\tilde{p} \in \{p, \neg p, ?p\}$	$Y \xrightarrow[\colorbox{lightgray}{$\langle \aleph\aleph_1{:}m,\, \beta\star\gamma\rangle$}]{\langle \aleph:-v,\, \beta\tilde{p}\rangle} Z \xleftarrow{\langle \aleph_1:-w,\, \gamma\rangle} P?$

$v > 0, w > 0, v' \geq 0, 0 < x \leq x' \leq y' \leq y, -\infty < u < \infty, Z, A, C, X, Y \in \mathcal{T}$; C is contingent; $P? \in \mathcal{OT}$; each of $\aleph$ and $\aleph_1$ is a conjunction of one or more upper-case names of contingent nodes, possibly empty; each of $\daleth$ and $\daleth_1$ is a conjunction of one or more lower-case names of contingent nodes, possibly empty.

A.2. Proof of Lemmas 2 and 3

Since the proofs of Lemmas 2 and 3 are very similar, we only prove Lemma 3.

Proof. The thesis to prove is that it is sufficient that the lower guard of a prototypal link is smaller, at least the path contingent span $\mathrm{cont}_S(X)$, to guarantee that any constraint derived from the prototypal link is satisfied in the network.

By definition, the lower guard of a prototypal link is always the minimum value among all possible conjunct-lower-case constraints and all upper-bound durations of all possible scenarios.

The thesis says that if the upper-bound of a prototypal link is restricted to v^*—where $\mathrm{uG}_S(X) \le v^* < y$—then the lower guard has to be updated to $\min\{\mathrm{lG}_S(X), v^* - \mathrm{cont}_S(X)\}$.

If the value of $\mathrm{lG}_S(X)$ is smaller than $v^* - \mathrm{cont}_S(X)$, it means that there is a stronger constraint in the network than the presence of guarded links.

Here we want to show that, on the contrary, if $\mathrm{lG}_S(X)$ is greater than $v^* - \mathrm{cont}_S(X)$, it is sufficient to lower $\mathrm{lG}_T(X)$ to $v^* - \mathrm{cont}_S(X)$ to guarantee that it is possible to execute the network without constraint violations. The proof is given showing that $v^* - \mathrm{cont}_S(X)$ is a safe execution timepoint for X. In particular, we show, by induction, that if $v^* = \mathrm{uG}_S(X)$ (the minimum possible v^*), then X can be executed at $\mathrm{lG}_T(X) = \min\{\mathrm{lG}_S(X), \mathrm{uG}_S(X) - \mathrm{cont}_S(X)\}$.

Since an FTNU is DC if its core is DC [21], in the following we restrict the attention to FTNUs where guarded links have outer bounds equal to their core bounds.

Base. The network comprises only two nodes, Z and X. There are two cases.

Case requirement link.

If the constraint between Z and X is the requirement link $Z\xrightarrow{[x,y]}X$, then the prototypal link representing the network has $\mathrm{lG}_S(X) = y$, $\mathrm{uG}_S(X) = x$, and the $\mathrm{cont}_S(X) = 0$, i.e., $Z\xRightarrow{[[x,y][x,y]]}X$.

If the upper bound is set to $\mathrm{uG}_S(X) = x$, then $\mathrm{lG}_T(X)$ has to be set to x, the only possible value, i.e., the prototypal link becomes $Z\xRightarrow{[[x,x][x,x]]}X$. The proposed expression $\min\{\mathrm{lG}_S(X), \mathrm{uG}_S(X) - \mathrm{cont}_S(X)\} = \min\{y, x-0\} = x$ determines it.

Case guarded link.

If the constraint between Z and X is a guarded link $Z\xRightarrow{[[x,x'][y',y]]}X$, then the prototypal link representing the network is the same, and the $\mathrm{cont}_S(X) = \mathrm{uG}_S(X) - \mathrm{lG}_S(X)$. In a guarded link, both guards cannot be changed by definition: $T \equiv S$. Thus, the proposed expression

$$\begin{aligned}\mathrm{lG}_T(X) &= \min\{\mathrm{lG}_S(X), \mathrm{uG}_S(X) - \mathrm{cont}_S(X)\}\\ &= \min\{\mathrm{lG}_S(X), \mathrm{uG}_S(X) - \mathrm{uG}_S(X) + \mathrm{lG}_S(X)\}\\ &= \mathrm{lG}_S(X)\end{aligned}$$

determines the right value.

Inductive step. Let us assume that the induction hypothesis holds for Z and a set of timepoints $Y_0, \dots, Y_k$, and that such timepoints have been executed at times $\psi(\mathrm{Z}) = 0, \psi(Y_0), \dots, \psi(Y_k)$. Let ℓ the current partial scenario, i.e., the propositional label composed by the true literals associated with the already executed observation timepoints, and that the timepoint X can be executed, i.e., all predecessor timepoints of X have been already executed. Its execution time has to be $\psi(X) \le \mathrm{uG}_S(X)$ by hypothesis. From the definition of conjunct-lower-case constraint, used for determining the $\mathrm{lG}_S(X)$, it is possible to derived that

$$\mathrm{lG}_T(X) = \min_{Y_i}\begin{cases}\mathrm{lG}_T(Y_i) + y_{i_T} & \text{if } Y_i\xrightarrow{[x_{i_T},\, y_{i_T}],\, \ell'}X,\ \ell' \text{ entails } \ell\\ \mathrm{lG}_T(Y_i) + x'_{i_T} & \text{if } Y_i\xRightarrow{[[x'_{i_T},\, x'_{i_T}][y'_{i_T},\, y'_{i_T}]]}X\end{cases}$$

where $Y_i, i = 1, \dots, k$, are the executed timepoints, and bounds with i_T subscript are bounds determined by the DC checking algorithm after the set of the upper bound $v^* = \mathrm{uG}_S(X)$ between Z and X in T.

By definition, $x_{i_S} \le x_{i_T}$, and, by hypothesis, $y'_{i_S} = y'_{i_T}$. Moreover, it is possible to show that $\mathrm{cont}_S() \equiv \mathrm{cont}_T()$ [47]. We excluded Z because it has no lower/higher guards. Applying the inductive hypothesis and the above equivalences, and considering the most significative case where the $\mathrm{lG}_T(Y_i) = \min\{\mathrm{lG}_S(Y_i), \mathrm{uG}_S(Y_i) - \mathrm{cont}_S(Y_i)\} = \mathrm{uG}_S(Y_i) - \mathrm{cont}_S(Y_i)$. It holds that

$$\mathrm{lG}_T(X) \ge \min_{Y_i}\begin{cases}\mathrm{uG}_S(Y_i) - \mathrm{cont}_S(Y_i) + y_{i_T} + x_{i_S} - x_{i_T} & \text{if } Y_i\xrightarrow{[x_{i_T},\, y_{i_T}],\, \ell'}X, \text{ and}\\ & \ell' \text{ entails } \ell\\ \mathrm{uG}_S(Y_i) - \mathrm{cont}_S(Y_i) + x'_{i_T} + y'_{i_S} - y'_{i_T} & \text{if } Y_i\xRightarrow{[[x'_{i_T},\, x'_{i_T}][y'_{i_T},\, y'_{i_T}]]}X\end{cases}$$

Now, in the case of the requirement link, it holds that

- $\mathrm{uG}_S(Y_i) + x_{i_S} \ge \mathrm{uG}_S(X)$ by definition of conjunct-upper-case constraint
- $\mathrm{cont}_T(Y_i) + (x_{i_T} - y_{i_T}) \le \mathrm{cont}_T(X)$ by definition of labeled-path-contingency span.

In the case of guarded link, analogously, it holds that $\mathrm{uG}_S(Y_i) + y'_{i_S} \ge \mathrm{uG}_S(X)$, and $\mathrm{cont}_T(Y_i) + y'_{i_T} - x'_{i_T} \le \mathrm{cont}_T(X) = \mathrm{cont}_S(X)$.
Therefore,

$$\mathrm{lG}_T(X) \ge \begin{cases}\mathrm{uG}_S(X) - \mathrm{cont}_S(X) & \text{if } Y_i\xrightarrow{[x_{i_T},\, y_{i_T}],\, \ell'}X,\ \ell' \text{ entails } \ell\\ \mathrm{uG}_S(X) - \mathrm{cont}_S(X) & \text{if } Y_i\xRightarrow{[[x'_{i_T},\, x'_{i_T}][y'_{i_T},\, y'_{i_T}]]}X\end{cases}$$

The choice $\mathrm{lG}_T(X) = \mathrm{uG}_S(X) - \mathrm{cont}_S(X)$ is the conservative one. □

References

[1] Business Process Model and Notation (BPMN), Object Management Group, 2014, version 2.0.2, https://www.omg.org/spec/BPMN/2.0.2/PDF, accessed 7 2021.

[2] M. Reichert, B. Weber, Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies, Springer, 2012, http://dx.doi.org/10.1007/978-3-642-30409-5.

[3] J. Vanhatalo, H. Völzer, J. Koehler, The refined process structure tree, Data Knowl. Eng. 68 (9) (2009) 793–818, http://dx.doi.org/10.1016/j.datak.2009.02.015.

[4] C. Combi, R. Posenato, Controllability in temporal conceptual workflow schemata, in: Business Process Management, 7th International Conference, BPM 2009, in: LNCS, vol. 5701, Springer, 2009, pp. 64–79, http://dx.doi.org/10.1007/978-3-642-03848-8_6.

[5] J. Eder, E. Panagos, M. Rabinovich, Time Constraints in Workflow Systems, Springer, 2013, pp. 191–205, http://dx.doi.org/10.1007/978-3-642-36926-1_15, Ch. 15.

[6] A. Lanz, R. Posenato, C. Combi, M. Reichert, Controllability of time-aware processes at run time, in: On the Move To Meaningful Internet Systems (OTM-2013), in: LNCS, vol. 8185, Springer, 2013, pp. 39–56, http://dx.doi.org/10.1007/978-3-642-41030-7_4.

[7] A. Lanz, B. Weber, M. Reichert, Time patterns for process-aware information systems, Requir. Eng. 19 (2) (2014) 113–141, http://dx.doi.org/10.1007/s00766-012-0162-3.

[8] 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. Syst. Man, Cybern. Syst. 44 (9) (2014) 1182–1203, http://dx.doi.org/10.1109/TSMC.2014.2300055.

[9] R. Posenato, A. Lanz, C. Combi, M. Reichert, Managing time-awareness in modularized processes, Softw. Syst. Model. 18 (2) (2019) 1135–1154, http://dx.doi.org/10.1007/s10270-017-0643-4.

[10] Y. Du, B. Yang, H. Hu, Incremental analysis of temporal constraints for concurrent workflow processes with dynamic changes, IEEE Trans. Ind. Inform. 15 (5) (2019) 2617–2627, http://dx.doi.org/10.1109/TII.2018.2868810.

[11] M. Franceschetti, J. Eder, Determining temporal agreements in cross-organizational business processes, Inform. and Comput. 281 (2021) 104792, http://dx.doi.org/10.1016/j.ic.2021.104792.

[12] H. Luo, X. Liu, J. Liu, Y. Yang, J.C. Grundy, Runtime verification of business cloud workflow temporal conformance, IEEE Trans. Serv. Comput. 15 (2) (2022) 833–846, http://dx.doi.org/10.1109/TSC.2019.2962666.

[13] A. Lanz, R. Posenato, C. Combi, M. Reichert, Controlling time-awareness in modularized processes, in: Enterprise, Business-Process and Information Systems Modeling. BPMDS 2016, EMMSAD 2016, in: Lecture Notes in Business Information Processing, vol. 248, Sprin, 2016, pp. 157–172, http://dx.doi.org/10.1007/978-3-319-39429-9_11.

[14] R. Posenato, C. Combi, Adding flexibility to uncertainty: Flexible simple Temporal Networks with Uncertainty (FTNU), Inform. Sci. 584 (2022) 784–807, http://dx.doi.org/10.1016/j.ins.2021.10.008.

[15] L. Hunsberger, R. Posenato, C. Combi, The dynamic controllability of conditional STNs with uncertainty, in: Work. on Planning and Plan Execution for Real-World Systems (PlanEx) At ICAPS-2012, 2012, pp. 1–8, URL http://arxiv.org/abs/1212.2005.

[16] L. Hunsberger, R. Posenato, Sound-and-complete algorithms for checking the dynamic controllability of conditional simple temporal networks with uncertainty, in: 25th Int. Symp. on Temporal Representation and Reasoning (TIME-2018), in: LIPIcs, vol. 120, 2018, pp. 14:1–14:17, http://dx.doi.org/10.4230/LIPIcs.TIME.2018.14.

[17] A. Lanz, R. Posenato, C. Combi, M. Reichert, Simple temporal networks with partially shrinkable uncertainty, in: 6th International Conference on Agents and Artificial Intelligence (ICAART 2015), Vol. 2, 2015, pp. 370–381, http://dx.doi.org/10.5220/0005200903700381.

[18] A. Cimatti, L. Hunsberger, A. Micheli, R. Posenato, M. Roveri, Dynamic controllability via timed game automata, Acta Inform. 53 (6–8) (2016) 681–722, http://dx.doi.org/10.1007/s00236-016-0257-2.

[19] A. Cimatti, L. Hunsberger, A. Micheli, R. Posenato, M. Roveri, Sound and complete algorithms for checking the dynamic controllability of temporal networks with uncertainty, disjunction and observation, in: A. Cesta, C. Combi, F. Laroussinie (Eds.), 21st Int. Symp. on Temporal Representation and Reasoning (TIME-2014), 2014, pp. 27–36, http://dx.doi.org/10.1109/TIME.2014.21.

[20] L. Hunsberger, R. Posenato, C. Combi, A sound-and-complete propagation-based algorithm for checking the dynamic consistency of conditional simple temporal networks, in: F. Grandi, M. Lange, A. Lomuscio (Eds.), 22nd Int. Symp. on Temporal Representation and Reasoning (TIME-2015), 2015, pp. 4–18, http://dx.doi.org/10.1109/TIME.2015.26.

[21] C. Combi, R. Posenato, Extending conditional simple temporal networks with partially shrinkable uncertainty, in: N. Alechina, K. Nørvåg, W. Penczek (Eds.), 25th International Symposium on Temporal Representation and Reasoning (TIME 2018), in: Leibniz International Proceedings in Informatics (LIPIcs), vol. 120, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, pp. 9:1–9:15, http://dx.doi.org/10.4230/LIPIcs.TIME.2018.9.

[22] E.C. Jauch, B. Cucchiara, O. Adeoye, W. Meurer, J. Brice, Y.Y. Chan, N. Gentile, M.F. Hazinski, Part 11: adult stroke: 2010 American heart association guidelines for cardiopulmonary resuscitation and emergency cardiovascular care, Circulation 122 (18 Suppl 3) (2010) S818–828, http://dx.doi.org/10.1161/CIRCULATIONAHA.110.971044.

[23] P. Ramanujam, E. Castillo, E. Patel, G. Vilke, M.P. Wilson, J.V. Dunford, Prehospital transport time intervals for acute stroke patients, J. Emergency Med. 37 (1) (2009) 40–45, http://dx.doi.org/10.1016/j.jemermed.2007.11.092.

[24] H. Pichler, J. Eder, M. Ciglic, Modelling processes with time-dependent control structures, in: ER 2017, Springer, 2017, pp. 50–58, http://dx.doi.org/10.1007/978-3-319-69904-2_4.

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

[26] M. Dumas, L. García-Bañuelos, A. Polyvyanyy, Unraveling unstructured process models, in: Business Process Modeling Notation - Second International Workshop, BPMN 2010, Proceedings, in: LNBIP, vol. 67, Springer, Potsdam, Germany, 2010, pp. 1–7, http://dx.doi.org/10.1007/978-3-642-16298-5_1.

[27] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction To Algorithms, third ed., MIT Press, 2009, URL http://mitpress.mit.edu/books/introduction-algorithms.

[28] R. Posenato, CSTNU Tool: A Java library for checking temporal networks, SoftwareX 17 (2022) 100905, http://dx.doi.org/10.1016/j.softx.2021.100905.

[29] R. Dechter, I. Meiri, J. Pearl, Temporal constraint networks, Artificial Intelligence 49 (1–3) (1991) 61–95, http://dx.doi.org/10.1016/0004-3702(91)90006-6.

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

[31] P. Morris, N. Muscettola, T. Vidal, Dynamic control of plans with temporal uncertainty, in: IJCAI 2001: Proc. of the 17th International Joint Conference on Artificial Intelligence, Vol. 1, 2001, pp. 494–499.

[32] L. Hunsberger, R. Posenato, Speeding up the RUL^{-} dynamic-controllability-checking algorithm for simple temporal networks with uncertainty, in: 36th AAAI Conference on Artificial Intelligence (AAAI-22), Vol. 36-9, AAAI Pres, 2022, pp. 9776–9785, http://dx.doi.org/10.1609/aaai.v36i9.21213.

[33] L. Hunsberger, R. Posenato, Simpler and faster algorithm for checking the dynamic consistency of conditional simple temporal networks, in: 26th Int. Joint Conf. on Artificial Intelligence, (IJCAI-2018), 2018, pp. 1324–1330, http://dx.doi.org/10.24963/ijcai.2018/184.

[34] L. Hunsberger, R. Posenato, Faster dynamic-consistency checking for conditional simple temporal networks, in: 30th Int. Conf. on Automated Planning and Scheduling, ICAPS 2020, Vol. 30, 2020, pp. 152–160, URL https://www.aaai.org/ojs/index.php/ICAPS/article/view/6656.

[35] C. Combi, L. Hunsberger, R. Posenato, An algorithm for checking the dynamic controllability of a conditional simple temporal network with uncertainty - revisited, in: Agents and Artificial Intelligence, in: Communications in Computer and Information Science (CCIS), vol. 449, Springer, 2014, pp. 314–331, http://dx.doi.org/10.1007/978-3-662-44440-5_19.

[36] A. Lanz, M. Reichert, Enabling time-aware process support with the atapis toolset, in: BPM Demo Sessions 2014, in: CEUR Workshop Proceedings, vol. 1295, 2014, pp. 41–45.

[37] L. Hunsberger, R. Posenato, Dynamic Controllability Checking for Conditional Simple Temporal Networks with Uncertainty: New Sound-and-Complete Algorithms Based on Constraint Propagation, Vol. 105, Tech. Rep., Computer Science Department-University of Verona, 2018, URL http://hdl.handle.net/11562/977720.

[38] C. Combi, R. Posenato, Towards temporal controllabilities for workflow schemata, in: 17th Intern. Symp. on Temporal Representation and Reasoning (TIME-2010), 2010, pp. 129–136, http://dx.doi.org/10.1109/TIME.2010.17.

[39] J. Eder, E. Panagos, M. Rabinovich, Workflow time management revisited, in: Seminal Contributions To Information Systems Engineering, Springer, 2013, pp. 207–213, http://dx.doi.org/10.1007/978-3-642-36926-1_16.

[40] T. Soetekouw, P. Grefen, O. Türetken, I.T.P. Vanderfeesten, A mass customization approach to business process modularization, in: 23rd Conference on Business Informatics, CBI 2021, Volume 2, IEEE, Bolzano, Italy, 2021, pp. 29–38, http://dx.doi.org/10.1109/CBI52690.2021.10052.

[41] C. Cappelli, F.M. Santoro, J.C.S. do Prado Leite, T.V. Batista, A.L. Medeiros, C. da Silva Correa Romeiro, Reflections on the modularity of business process models: The case for introducing the aspect-oriented paradigm, Bus. Process. Manag. J. 16 (4) (2010) 662–687, http://dx.doi.org/10.1108/14637151011065955.

[42] R. Conforti, M. Dumas, L. García-Bañuelos, M.L. Rosa, Beyond tasks and gateways: Discovering BPMN models with subprocesses, boundary events and activity markers, in: Business Process Management - 12th International Conference, BPM 2014, in: LNCS, vol. 8659, Springer, 2014, pp. 101–117, http://dx.doi.org/10.1007/978-3-319-10172-9_7.

[43] A. Lanz, M. Reichert, B. Weber, A Formal Semantics of Time Patterns for Process-Aware Information Systems, Technical Report UIB-2013-02, University of Ulm, 2013, URL http://dbis.eprints.uni-ulm.de/894/.

[44] C. Combi, B. Oliboni, F. Zerbato, A modular approach to the specification and management of time duration constraints in BPMN, Inf. Syst. 84 (2019) 111–144, http://dx.doi.org/10.1016/j.is.2019.04.010.

[45] H. Norman, S. Debois, T. Slaats, T.T. Hildebrandt, Zoom and enhance: Action refinement via subprocesses in timed declarative processes, in: A. Polyvyanyy, M.T. Wynn, A.V. Looy, M. Reichert (Eds.), Business Process Management - 19th International Conference, BPM 2021, Proceedings, in: LNCS, vol. 12875, Springer, Rome, Italy, 2021, pp. 161–178, http://dx.doi.org/10.1007/978-3-030-85469-0_12.

[46] M. Käppel, L. Ackermann, S. Schönig, S. Jablonski, Language-independent look-ahead for checking multi-perspective declarative process models, Softw. Syst. Model. 20 (5) (2021) 1379–1401, http://dx.doi.org/10.1007/s10270-020-00857-8.

[47] A. Lanz, R. Posenato, C. Combi, M. Reichert, Controlling Time-Awareness in Modularized Processes (Extended Version), Technical Report UIB-2015-01, Ulm University, 2015, URL http://dbis.eprints.uni-ulm.de/1133/.