Journal on Data Semantics (2021) 10:19–39
https://doi.org/10.1007/s13740-021-00119-w

ORIGINAL ARTICLE

Resource Controllability of Business Processes Under Conditional Uncertainty

Matteo Zavatteri[1] · **Carlo Combi**[1] · **Luca Viganò**[2]

Received: 3 February 2020 / Revised: 14 December 2020 / Accepted: 21 January 2021 / Published online: 5 March 2021

Abstract
A current research problem in the area of business process management deals with the specification and checking of constraints on resources (e.g., users, agents, autonomous systems, etc.) allowed to be committed for the execution of specific tasks. Indeed, in many real-world situations, role assignments are not enough to assign tasks to the suitable resources. It could be the case that further requirements need to be specified and satisfied. As an example, one would like to avoid that employees that are relatives are assigned to a set of critical tasks in the same process in order to prevent fraud. The formal specification of a business process and its related access control constraints is obtained through a decoration of a classic business process with roles, users, and constraints on their commitment. As a result, such a process specifies a set of tasks that need to be executed by authorized users with respect to some partial order in a way that all authorization constraints are satisfied. Controllability refers in this case to the capability of executing the process satisfying all these constraints, even when some process components, e.g., gateway conditions, can only be observed, but not decided, by the process engine responsible of the execution. In this paper, we propose *conditional constraint networks with decisions (CCNDs)* as a model to encode business processes that involve access control and conditional branches that may be both controllable and uncontrollable. We define weak, strong, and dynamic controllability of CCNDs as two-player games, classify their computational complexity, and discuss strategy synthesis algorithms. We provide an encoding from the business processes we consider here into CCNDs to exploit off-the-shelf their strategy synthesis algorithms. We introduce Zeta, a tool for checking controllability of CCNDs, synthesizing execution strategies, and executing controllable CCNDs, by also supporting user interactivity. We use Zeta to compare with the previous research, provide a new experimental evaluation for CCNDs, and discuss limitations.

Keywords Access controlled processes · Resource allocation under uncertainty · Online planning · Resource controllability · CCND · Zeta · AI-based security · Business process compliance under uncertainty

1 Introduction

Business process management (BPM) is a scientific and technological area that focuses on issues and solutions related to the support and coordination of complex and interrelated activities within one or many organizations [3,18,46]. BPM has been defined in [47] as "Supporting business processes using methods, techniques, and software to design, enact, control, and analyze operational processes involving humans, organizations, applications, documents and other sources of information."

The terms "business process" (BP) and "workflow" (WF) have been often used as synonyms, but BP has also been considered to be a term with a wider meaning than that of WF. For example, the Workflow Management Coalition (WfMC) defines WF as the automation of a *business process*, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules [47]. Thus, a *WorkFlow Management System (WFMS)* is defined as a system that defines, creates and manages the execution of WFs through the use of

✉ Matteo Zavatteri
matteo.zavatteri@univr.it

Carlo Combi
carlo.combi@univr.it

Luca Viganò
luca.vigano@kcl.ac.uk

[1] Department of Computer Science, University of Verona, Verona, Italy

[2] Department of Informatics, King's College London, London, UK

software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of Information-Technology (IT) tools and applications. In the following, we will use the term WF, to underline the fact that the considered BP has been suitably formalized for automation and for checking specific properties.

A current research problem in this area is the management of constraints on users allowed to execute specific tasks [36,39]. Indeed, in many real-world situations, it could be required that, within the execution of a single business process, users assigned to different tasks have the right role and, moreover, they satisfy some further conditions. As an example, to prevent a possible conflict of interest, one typically wants to avoid that married people work in the same process, even though with different roles and in different tasks. The need of managing roles and access control in the execution of tasks of a business process is also related to security issues, often in a context-depending scenario, where constraints on roles and users have to be guaranteed to have suitably authorized and safe processes [30,42]. Recently, it has been highlighted that security issues, often consisting of access control for the different tasks together with task-related data protection, have to be considered in depth, even with respect to some possible conflicts with other business process requirements (e.g., preserving the privacy without causing possible general risks) [38]. Access control requirements have been also highlighted by software companies involved in process-oriented information systems [4,5].

In this direction, an *access controlled WF (ACWF)* augments a WF by adding users authorized for tasks and authorization constraints saying which combinations of assignments of tasks to users are permitted. When a WF specifies a set of temporal, conditional, or authorization constraints and all of its components are under control we simply deal with a *satisfiability problem* asking us to find a fixed solution satisfying all constraints. Instead, when some component is out of control (e.g., task durations or conditional constraints) we deal with a *controllability problem*, where the synthesis of a fixed execution plan is not enough.

Controllability implies the existence of a *strategy* to operate (possibly differently) on the part under our control depending on the behavior of some uncontrollable events that we will only be able to observe while executing. This means that, depending on how this uncontrollable part behaves, we might schedule the same tasks at different times or commit different users for the same tasks.

Controllability of *temporal workflows (TWFs)* (i.e., WFs augmented with task duration constraints, delays and deadlines) addresses uncontrollable task durations and conditional uncertainty (i.e., the uncontrollable choice of the WF path to take at runtime). A possible way to check controllability of TWFs is by encoding the TWF into a corresponding temporal network to boil down the controllability of the TWF to that of the temporal network for which controllability checking algorithms exist (e.g., [7–9,19,23,37,49,52,53,60]). Like TWFs, in ACWFs conditional uncertainty models the uncontrollable choice of the WF path to take during execution. For instance, when a patient comes to the ER, the therapies/interventions he has to undergo are not known a priori but they are established by a physician, *while* the WF is being executed. Since the result of this choice discriminates what tasks have, or have not, to be executed, and which users are committed for the same tasks, the system must be able to complete the WF by executing all relevant tasks and satisfying all relevant authorization constraints regardless of the result of (any combination) of uncontrollable conditions. When the assignment of tasks to users is generated while the WF executes we must never backtrack. In the real world, this means that we must avoid situations in which, if a patient needs a specialized intervention, no surgeon is available because we chose to assign the "wrong" surgeon to some previous general task.

In [51], an approach to address controllability of ACWFs is provided. That approach maps WF paths to *constraint networks* (*CNs*, [14]), relies on *directional consistency* to guarantee no backtracking when generating any solution to the underlying constraints satisfaction problem [17], and reasons on the intersection of the common parts of the WF paths to achieve a dynamic execution of the ACWFs. However, dynamic controllability of ACWFs also depends on how the components of the ACWF are ordered [59,61]. In [51], the designer encodes manually an ACWF into a CN meaning that it is up to him to choose a suitable total order for each WF path such that these WF paths can be intersected in their common parts to rule out assignments of tasks to users that never satisfy any solution. If the designer chooses the wrong order (even for one WF path only), either the controllability algorithm is not applicable or a controllable ACWF might be classified as uncontrollable.

With respect to these previous contributions, in this paper we focus on a wider scenario, and deal with the validation and runtime enactment of business processes with access control and conditional branches that may be either controllable or uncontrollable. This paper thus revises and extends our previous work in [54]. The main novelties may be summarized as follows.

- We introduce a new kind of network, namely conditional constraint network with decisions (CCND), to represent both controllable and uncontrollable choices together with relation constraints.
- We propose a mapping from an extended fragment of BPMN (business process model and notation, [3]) to CCND constructs.

– We define and discuss weak, strong, and dynamic controllability for CCNDs and show how to check them on extended BPMN processes, by also allowing for the representation of roles, agents, and related constraints.
– We describe and discuss an improved tool for checking controllabilities of CCNDs, together with an extended experimental evaluation that also compares with the software tool we previously used in [54].

In Sect. 2, we extend the motivating example used in [54] by adding an inclusive OR block. In Sect. 3, we define conditional constraint networks with decisions by extending constraint networks under conditional uncertainty (CNCUs) used in [54] with decision variables to encode controllable conditional branches and a labeled partial order relation. Section 4 provides an encoding from business processes we consider here into CCNDs. The extension we provide here augments that in [54] by also considering controllable conditional branches, inclusive OR blocks, and further mutual exclusivity constraints on task execution. In Sect. 5, we define weak, strong, and dynamic controllability of CCNDs, classify their computational complexity, and discuss strategy synthesis algorithms. In Sect. 6, we provide a new version of ZETA, completely rewritten. We use ZETA to compare against a previous experimental evaluation for CNCUs and also provide a new one for CCNDs. There, we briefly discuss some technical and non-technical limitations of our approach. In Sect. 7, we give an overview on related work on resource controllability, and finally in Sect. 8, we draw some conclusions.

2 Running Example

As a motivating example we will use throughout the paper, we consider a simplified loan origination process (LOP). Figure 1 depicts the process structure, according to the widely known BPMN notation [3], extended to graphically represent roles, resources, and the related constraints. The process consists of 10 tasks, 6 roles (`Clerk`, `Auditor`, `AMLOfficer`, `IRSOfficer`, `Manager`, and `System`), 6 users (`alice`, `bob`, `evie`, `kate`, `mike` and `ted`), and 1 automated agent (`server`). `Clerk` contains `alice` and `bob`, `Auditor` contains `bob` and `kate`, `AMLOfficer` contains `mike` only, `IRSOfficer` contains `evie` only, `Manager` contains `kate` and `ted`, whereas `System` contains `server` only.

A `Clerk` starts the process by processing a loan request (`ProcR`). After that, the flow of the execution splits by entering an unconditional parallel block. In this block, an `Auditor` checks the financial records of the customer (`CheckFR`) and at the same time another further verification takes place depending on if the amount of money requested is huge or not. If `hugeA?` is true, then an `AMLOfficer` carries out an anti-money laundering assessment (`AntiML`). If `hugeA?` is false, then an `IRSOfficer` carries out a simple tax fraud assessment (`TaxFA`). The `Auditor` who executes `CheckFR` must be different from the `Clerk` who executed `ProcR`, (as some users, e.g., `bob`, might belong to both roles) and must also not be a relative of the `AMLOfficer` who executed `AntiML` nor of the `IRSOfficer` who executed `TaxFA` (it is not necessary for them to be different since the bank requires `AMLOfficers` and `IRSOfficers` to be external disjoint consultants). After that, both the internal conditional block and the external parallel one (in this order) complete and the execution flow enters a new conditional block to carry out the final tasks depending on if the loan has been approved or not. If `app?` is true, then a `Manager` prepares the contract (`PrepC`) and another one, who must be different from the first, signs it (`Sign`). If `app?` is false, then the same `Clerk` who executed the initial `ProcR` rejects the request (`Reject`).

Finally, the process goes through an inclusive OR block where a notification is sent. Specifically, an agent of `System` sends an email (`EmailN`) plus either a text notification (`TextN`) or a mobile app notification (`MobN`). This last either-or part is up to the system. After that, the process completes.

As the overall goal is that of executing this process always satisfying the precedence, the authorization constraints, and the mutual-exclusion constraints between tasks no matter which execution path we go through, we would like to be able to establish at design time and then manage at runtime different controllability-related features.

3 Conditional Constraint Networks with Decisions

In this section, we introduce a new formalism suitable to encode a more expressive fragment of BPMN with respect to our previous work in [54]. We deal with both controllable and uncontrollable parts and model the arising controllability problems as two-player games between Controller (operating on controllable parts) and Nature (operating on uncontrollable parts).

Given a set $\mathcal{P}$ of Boolean propositions, a *label* $\ell = \lambda_1 \dots \lambda_n$ is any finite conjunction of literals λ_i, where a literal is either a proposition p (positive literal) or its negation $\neg p$ (negative literal). The *empty label* is denoted by $\boxdot$. The *label universe of* $\mathcal{P}$, denoted by $\mathcal{P}^*$, is the set of all possible (consistent) labels drawn from $\mathcal{P}$, e.g., if $\mathcal{P} = \{p, q\}$, then $\mathcal{P}^* = \{\boxdot, p, q, \neg p, \neg q, p \wedge q, p \wedge \neg q, \neg p \wedge q, \neg p \wedge \neg q\}$. Two labels $\ell_1, \ell_2 \in \mathcal{P}^*$ are *consistent* if and only if their conjunction $\ell_1 \wedge \ell_2$ is satisfiable. A label ℓ_1 *entails* a label ℓ_2 (written $\ell_1 \Rightarrow \ell_2$) if and only if all literals in ℓ_2 appear

in ℓ_1 too (i.e., if ℓ_1 is more *specific* than ℓ_2). For instance, if $\ell_1 = p \wedge \neg q$ and $\ell_2 = p$, then ℓ_1 and ℓ_2 are consistent since $p \wedge \neg q \wedge p$ is satisfiable, and ℓ_1 entails ℓ_2 since $p \wedge \neg q \Rightarrow p$.

Definition 1 A *conditional constraint network with decisions (CCND)* is a tuple $\langle \mathcal{X}, \mathcal{V}, D, \mathcal{O}, \mathcal{D}, \mathcal{P}, O, L, \preceq, \mathcal{C} \rangle$, where:

1. $\mathcal{X} = \{X_1, \ldots, X_n\}$ is a finite non-empty set of variables.
2. $\mathcal{V} = \{\mathtt{a}, \mathtt{b}, \ldots\}$ is a finite non-empty set of discrete values.
3. $D \subseteq \mathcal{X} \times \mathcal{V}$ is the *domain relation* such that $(X, \mathtt{v}) \in D$ means that X can be assigned $\mathtt{v}$. We abuse notation and shorten the *domain* of any $X \in \mathcal{X}$ as $D(X) = \{\mathtt{v} \mid (X, \mathtt{v}) \in D\}$ and assume that all $D(X)$ are non-empty.
4. $\mathcal{O} \subseteq \mathcal{X} = \{P?, Q?, \ldots\}$ is a set of *observation variables* and $\mathcal{D} \subseteq \mathcal{X} = \{A!, B!, \ldots\}$ is a set of *decision variables*. Moreover, $\mathcal{O}$ and $\mathcal{D}$ are disjoint sets.
5. $\mathcal{P} = \{p, q, \ldots\}$ is a set of Boolean *propositions* whose truth values are all initially unknown.
6. $O : \mathcal{P} \rightleftharpoons \mathcal{O} \cup \mathcal{D}$ is a bijection assigning either an observation or a decision variable to each proposition. When an observation variable $P?$ is assigned a value $\mathtt{v} \in D(P?)$, the truth value of p is set by Nature and no longer changes. Instead, when a decision variable $A!$ is assigned a value $\mathtt{v} \in D(A!)$, the truth value of a is set by Controller.
7. $L : \mathcal{X} \to \mathcal{P}^*$ is a mapping assigning a label ℓ to each variable.
8. $\preceq \; \subseteq \mathcal{X} \times \mathcal{X} \times \mathcal{P}^*$ is a labeled precedence relation on the variables. We write $(X_1, X_2, \ell) \in \preceq$ (or more conveniently $(X_1 \preceq X_2, \ell)$) to express that if ℓ is true, then X_1 executes *before* X_2.
9. $\mathcal{C}$ is a finite set of *labeled relational constraints* of the form (R, ℓ), where $R = (S, T)$ is a relation with non-empty scope $S = \{X_j, \ldots, X_k\} \subseteq \mathcal{X}$ and set of tuples $T \subseteq D(X_j) \times \cdots \times D(X_k)$, whereas $\ell \in \mathcal{P}^*$.

A CCND is *well defined* if and only if the following properties hold.

(1) For each $X \in \mathcal{X}$, if a literal p (or $\neg p$) $\in L(X)$, then $L(X) \Rightarrow L(O(p))$ and $(O(p) \preceq X, L(X))$ (variable label honesty).
(2) For each constraint $((S, T), \ell) \in \mathcal{C}$, if a literal p (or $\neg p$) $\in L(X)$, then $\ell \Rightarrow L(O(p))$ (constraint label honesty). Furthermore, $\ell \Rightarrow L(X)$ for each $X \in S$ (constraint label coherence).
(3) For each $(X_1, X_2, \ell) \in \preceq$, if a literal p (or $\neg p$) $\in \ell$, then $\ell \Rightarrow L(O(p))$ (precedence label honesty). Moreover, $\ell \Rightarrow L(X_1)$ and $\ell \Rightarrow L(X_2)$. □

A constraint network under conditional uncertainty (CNCU, [55,59,61])—the model we used in the previous version of this paper—is equivalent to a CCND where $\mathcal{D} = \emptyset$, and $\ell = \boxdot$ for each $(X, Y, \ell) \in \preceq$.

We say that a variable, precedence, or relational constraint is *relevant* if and only if its label is true (once the involved propositions have been assigned).

A CCND is *binary* if all constraints $((S, T), \ell) \in \mathcal{C}$ are such that $|S| \leq 2$. We graphically represent a binary CCND as a *labeled constraint (multi)graph*, where each variable X is labeled by its label $L(X) \in \mathcal{P}^*$ and its domain $D(X) = \{\mathtt{v}_1, \ldots, \mathtt{v}_n\}$. Edges are of two kinds: *order edges* (directed labeled edges) and *constraint edges* (undirected labeled edges). An order edge $X_1 \to X_2$ labeled by ℓ models $(X_1 \preceq X_2, \ell)$. A constraint edge between X_1 and X_2 labeled by (R, ℓ) models $(R, \ell) \in \mathcal{C}$ where $R = (\{X_1, X_2\}, T)$. Many constraint edges may possibly be specified between the same pair of variables (or more conveniently, many labels on the same edge), as long as ℓ is different.

4 Encoding Processes into CCNDs

Table 1 provides an extension of the encoding from the fragment of BPMN discussed in [51,54] into CCNDs. Despite such an encoding is quite restrictive with respect to the number of components, it handles a number of process blocks that are crucial to many process instances. In this section, we follow a structured approach and deal with what in the BPM community are known as process blocks. A process block can be seen at a more theoretical level as a context-free grammar in which tasks and skips are terminal symbols and everything else is non-terminal. Before discussing our process blocks we spend a few words on access control.

A *role-based access control model* (*RBAC*, [41]) relies on the concept of *role* that acts as an interface between users and permissions. If a user changes his roles the security officer simply reassigns him to the new ones. An *RBAC model for a process* is a tuple

$$\langle \mathtt{Roles}, \mathtt{Users}, \mathtt{Tasks}, UA, TA \rangle$$

where $\mathtt{Roles} = \{r_1, \ldots, r_n\}$, $\mathtt{Users} = \{u_1, \ldots, u_m\}$ and $\mathtt{Tasks} = \{t_1, \ldots, t_k\}$ are finite sets of roles, users and tasks, respectively, and $UA \subseteq \mathtt{Users} \times \mathtt{Roles}$ and $TA \subseteq \mathtt{Tasks} \times \mathtt{Roles}$ are the user-role and task-role assignment relations, respectively. We write $users(t) = \{u \mid \exists r \in \mathtt{Roles}, (t, r) \in TA, (u, r) \in UA\}$ for the set of users authorized for t.

Table 1 (left column) provides the encoding from (acyclic) process blocks into CCNDs. Note that loops could be modeled by unfolding a maximum number of iterations where each iteration is modeled by a choice block (positive condition means iterate, negative one means stop). Tasks are labeled by a finite set of roles $\{r_1, \ldots, r_n\} \subseteq \mathtt{Roles}$ mean-

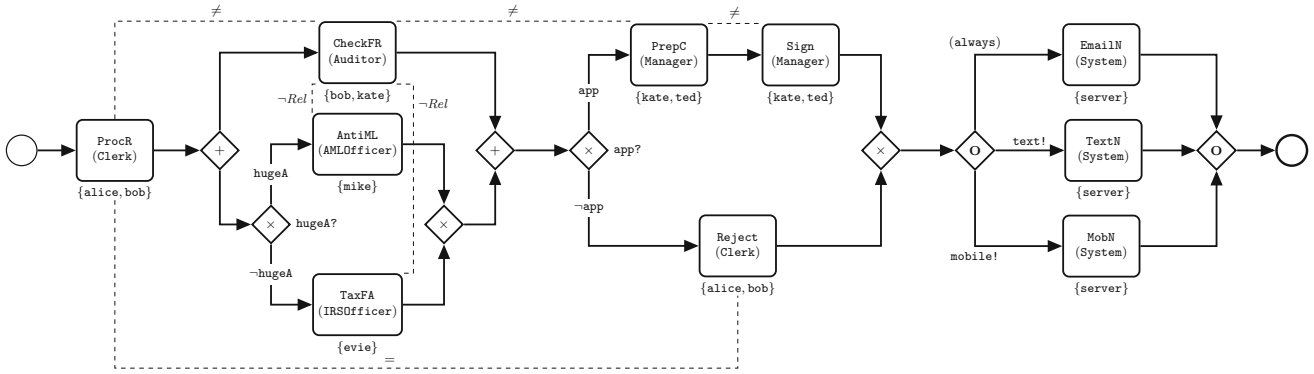
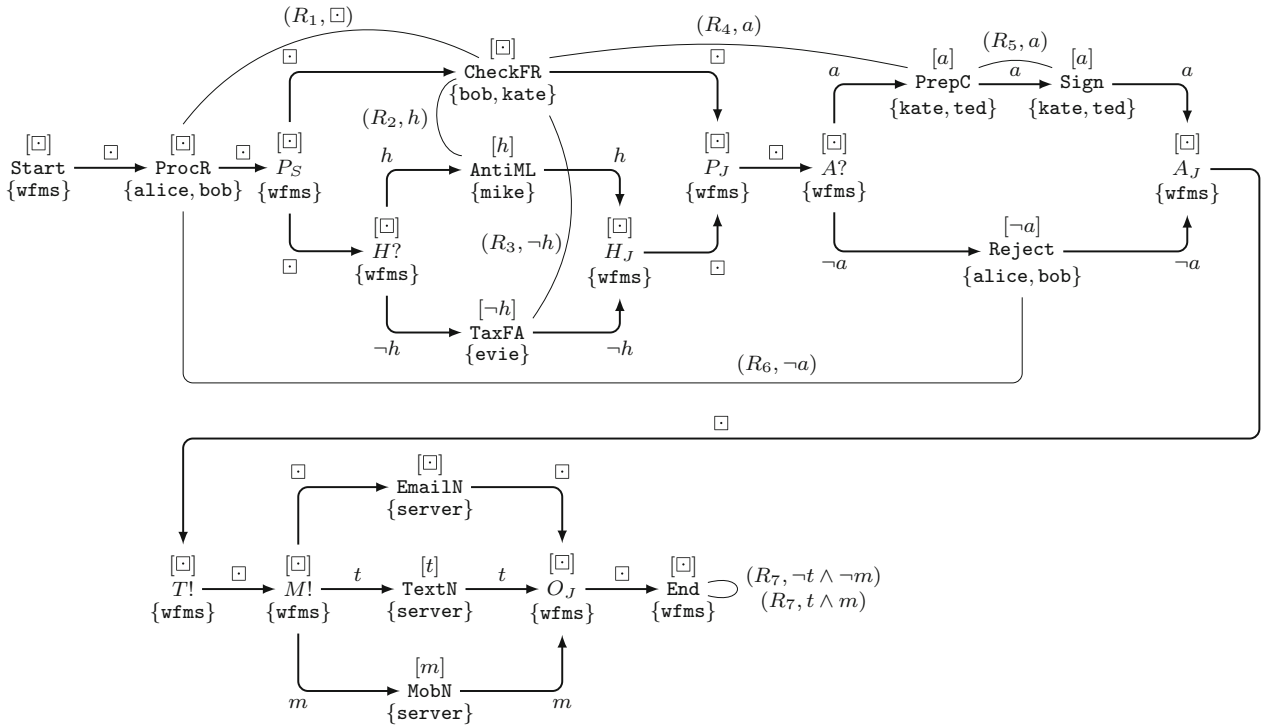


Fig. 1 Example of business process with access control in BPMN-like graphical style for a loan origination process. The BPMN notation has been extended in an intuitive way to represent roles (within round brackets in the task labels), resources (within braces below each task), and labeled dashed lines for constraints between the connected tasks. `bob` and `mike` are brothers. `kate` and `evie` are sisters. Email notifications are always sent in addition to either a text or a mobile app notification



(a) CCND encoding of the process in Figure 1.

ProcR	CheckFR
alice	bob
alice	kate
bob	kate

(b) R_1

CheckFR	AntiML
kate	mike

(c) R_2

CheckFR	TaxFA
bob	evie

(d) R_3

CheckFR	PrepC
bob	kate
bob	ted
kate	ted

(e) R_4

PrepC	Sign
kate	ted
ted	kate

(f) R_5

ProcR	Reject
alice	alice
bob	bob

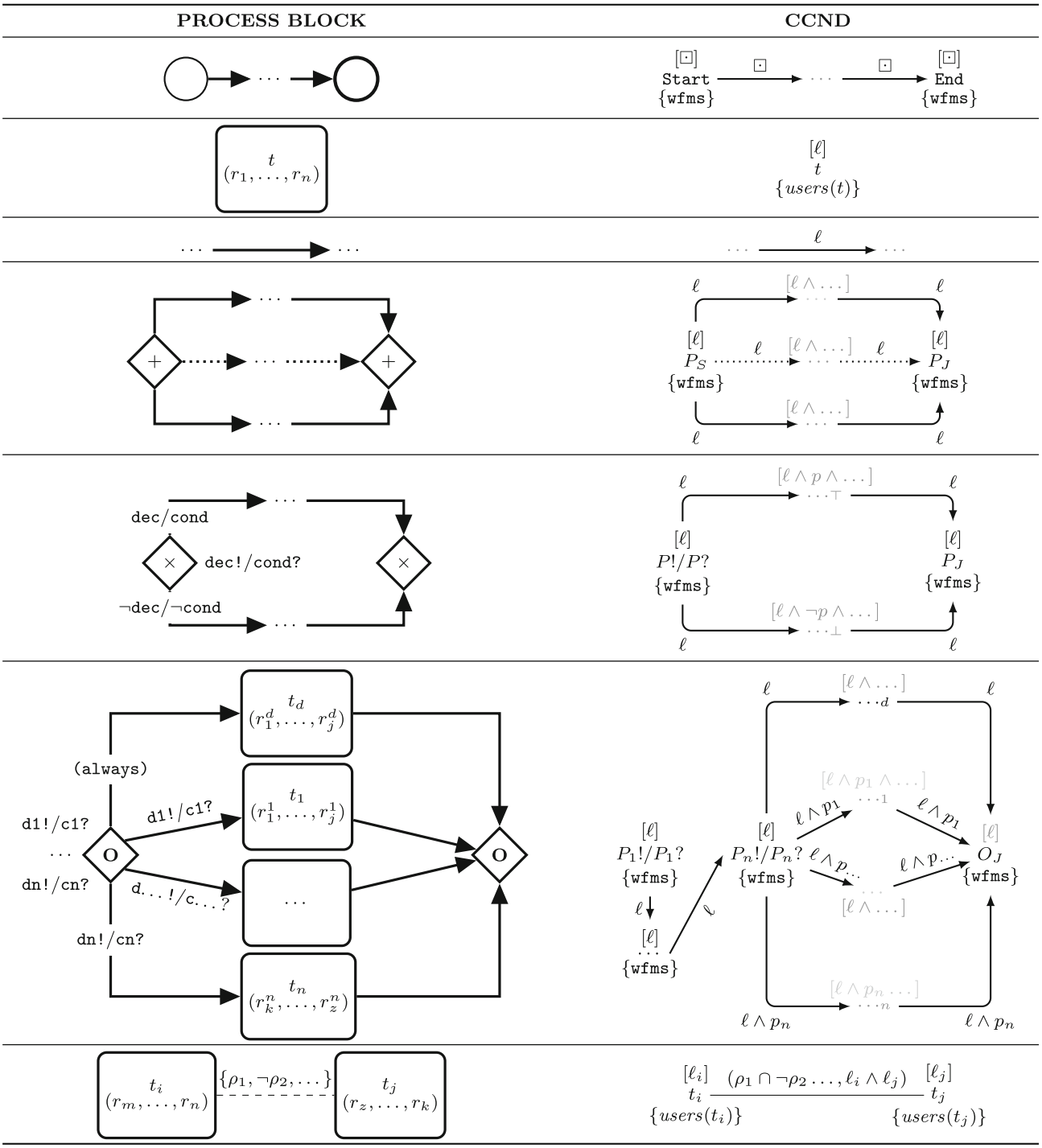
(g) R_6

End
$\emptyset$

(h) R_7

Fig. 2 **a** The CCND corresponding to BPMN process in Fig. 1; **b**–**h** the relational constraints

Table 1 Encoding process blocks into CCNDs



ing that $(t, r_1), \ldots, (t, r_n) \in TA$ (Table 1, 2nd row). Assigning roles to tasks models "who does what." Each mutual exclusive gateway is associated with a unique Boolean variable `dec!/cond?` whose truth value assignment may be controllable or not depending on if we desire to model a decision or a condition (Table 1, 5th row).

However, classical RBAC models fail to specify security policies at user level such as *separation of duties (SoD)* and *binding of duties (BoD)*.[1]

Authorization constraints address such an issue. Authorization constraints are defined for a subset of non-mutually exclusive tasks $t_1, \ldots, t_n$ and are formalized as a set of relations $\{\rho_1, \ldots, \rho_m\}$ each one defined over $\texttt{Users}^m$ (i.e., m-times the cross product of `Users`). Each relation may appear positive (ρ) or negative ($\neg\rho$). If $u_1 \in users(t_1)$, $u_2 \in users(t_2)$, ..., $u_n \in users(t_n)$ and $(u_1, u_2, \ldots, u_n)$ satisfies all $(\neg)\rho_i$ in the set, then any execution assigning t_1 to u_1, t_2 to u_2,..., and t_n to u_n *satisfies* the authorization constraint. Whenever an authorization constraint restricts to only two tasks we can easily draw it as an undirected dashed edge between them (Table 1, last row).

The start and end of a process are encoded as two variables `Start` and `End` occurring before and after all other variables, respectively. $L(\texttt{Start}) = L(\texttt{End}) = \boxdot$ since the start and the end of a process always occur. `wfms`—the workflow management system—is the unique authorized user for these variables. No constraint edge involves `Start` and `End` (Table 1, row 1). A task t having authorized roles $r_1, \ldots, r_n$ is encoded as a homonymous variable whose domain consists of the union of all users belonging to $r_1, \ldots, r_n$ authorized for t, i.e., $users(t)$, whereas its label contains the propositions modeling the Boolean variables associated with the mutual exclusive gateways according to the nesting level of the block in which the task appears (Table 1, row 2). All arcs regulating the control flow are encoded as order edges (Table 1, row 3). Parallel and conditional blocks are straightforwardly encoded mirroring the partial order of the process in the CCND. If the block is a parallel, a variable P_S models the parallel gateway, whereas a variable P_J models the corresponding join. $L(P_S) = L(P_J)$ according to the nesting level of the block in the process. All labels of the variables modeling the components inside the block in the process (if any) must entail $L(P_S)$ in the CCND (Fig. 1, row 4). If a block is a choice then a decision variable $P!$ or an observation variable $P?$ having associated a proposition p models the mutual exclusive gateway. In the first case, the choice of the branch to take is controllable, whereas in the second it is uncontrollable. P_E still models the corresponding join. Again, $L(P!/P?) = L(P_E)$ but this time as well as entailing $L(P!/P?)$, all labels of the variables modeling the components belonging to the true and false branch in the process (if any) are augmented with p or $\neg p$, respectively, in the CCND (Table 1, row 5). All variables modeling gateways are all executed by `wfms`.

An inclusive OR block consists of a gateway having n associated Boolean variables, $n+1$ outgoing branches, and a join. Each Boolean variable, which, again, may be a decision or a condition, is associated one-to-one to a specific branch. The execution flow goes through any of these branches if and only if the corresponding Boolean variable is true. Instead, the $(n+1)$-th branch—the default branch—is always executed. To encode an inclusive OR block into a CCND we replace the split gateway with a sequence of as many decision/observation variables as the number of conditions and decisions. After that, we connect the last of these variables to each branch (and coherently label the branch like the previous discussed blocks), and finally connect the end of all internal blocks to a variable O_J modeling the join connector (Table 1, row 6).

An authorization constraint between two non-mutually exclusive tasks is encoded as a constraint edge whose relation is the intersection of all relations appearing on the authorization constraint in the process block and the label is the conjunction of the labels of the variables modeling tasks (Table 1, row 7). Likewise, despite we do not show it in Table 1, an authorization constraint involving n non-mutually exclusive tasks is encoded into a n-ary relation in the CCND along with the conjunction of the labels of the corresponding variables.

Finally, further constraints on the combination of allowed decisions/conditions can be encoded into CCNDs. For example, the process in Fig. 1 requires that the end of the process exactly one between `text!` and `mobile!` is true. To encode it we proceed as follows. Let t and m be the corresponding propositions in the CCND. Let X be any variable in the CCND (it really does not matter which variable). We add an unsatisfiable constraint $((\{X\}, \emptyset), \neg t \wedge \neg m)$ to impose that at least one between t and m must be true, and another unsatisfiable constraint $((\{X\}, \emptyset), t \wedge m)$ to prevent that both cannot be true in the same execution.

Figure 2a represents the encoding of the process depicted in Fig. 1. We show the relational constraints encoding authorization constraints and mutual exclusivity of `text!` and `mobile!` in Fig. 2b, h.

Figure 2a shows a graph-based representation of the CCND $\mathcal{Z} = \langle \mathcal{X}, \mathcal{V}, D, \mathcal{O}, \mathcal{D}, \mathcal{P}, O, L, \preceq, \mathcal{C} \rangle$ in which:

- $\mathcal{X} := \{\texttt{Start}, \texttt{ProcR}, P_S, H?, \texttt{AntiML}, \texttt{TaxFA}, H_J, P_J, A?, \texttt{PrepC}, \texttt{Sign}, \texttt{Reject}, A_J, T!, M!, \texttt{EmailN}, \texttt{TextN}, \texttt{MobN}, O_J, \texttt{End}\}$
- $\mathcal{V} := \{\texttt{alice}, \texttt{bob}, \texttt{kate}, \texttt{evie}, \texttt{mike}, \texttt{ted}, \texttt{wfms}, \texttt{server}\}$

[1] SoD is a security policy saying that a subset of tasks must be carried out by different users, whereas BoD says that a subset of tasks must be carried out by the same user.

- $D(\texttt{Start}) = D(P_S) = D(H?) = D(H_J) = D(P_J) = D(A?) = D(A_J) = D(T!) = D(M!) = D(O_J) = D(\texttt{End}) := \{\texttt{wfms}\}$
- $D(\texttt{ProcR}) = D(\texttt{Reject}) := \{\texttt{alice}, \texttt{bob}\}$
- $D(\texttt{CheckFR}) := \{\texttt{bob}, \texttt{kate}\}, D(\texttt{AntiML}) := \{\texttt{mike}\}$, $D(\texttt{TaxFA}) := \{\texttt{evie}\}$, $D(\texttt{PrepC}) = D(\texttt{Sign}) := \{\texttt{kate}, \texttt{ted}\}, D(\texttt{EmailN}) = D(\texttt{TextN}) = D(\texttt{MobN}) := \{\texttt{server}\}, \mathcal{O} := \{H?, A?\}$,
- $\mathcal{D} := \{T!, M!\}, \mathcal{P} := \{h, a, t, m\}$
- $O(h) := H?, O(a) := A?, O(t) := T!, O(m) := M!$,
- $L(\texttt{AntiML}) := h, L(\texttt{TaxFA}) := \neg h$, $L(\texttt{PrepC}) = L(\texttt{Sign}) := a, L(\texttt{Reject}) := \neg a$, $L(\texttt{MobN}) := m$ and $L(\texttt{TextN}) := t, L(X) := \boxdot$ for all $X \in \mathcal{X} \setminus \{\texttt{AntiML}, \texttt{TaxFA}, \texttt{PrepC}, \texttt{Sign}, \texttt{Reject}, \texttt{TextN}, \texttt{MobN}\}$
- $(\texttt{Start} \preceq \texttt{ProcR}, \boxdot)$, $(\texttt{ProcR} \preceq P_S, \boxdot)$, $(P_S \preceq \texttt{CheckFR}, \boxdot)$, $(\texttt{CheckFR} \preceq P_J, \boxdot)$, $(P_S \preceq H, \boxdot)$, $(H \preceq \texttt{AntiML}, h)$, $(\texttt{AntiML} \preceq H_J, h)$, $(H? \preceq \texttt{TaxFA}, \neg h)$, $(\texttt{TaxFA} \preceq H_J, \neg h)$, $(H_J \preceq P_J, \boxdot)$, $(P_J \preceq A, \boxdot)$, $(A? \preceq \texttt{PrepC}, a)$, $(\texttt{PrepC} \preceq \texttt{Sign}, a)$, $(\texttt{Sign} \preceq A_J, a)$, $(A? \preceq \texttt{Reject}, \neg a)$, $(\texttt{Reject} \preceq A_J, \neg a)$, $(A_J \preceq T!, \boxdot)$, $(T! \preceq M!, \boxdot)$, $(M! \preceq \texttt{EmailN}, \boxdot)$, $(\texttt{EmailN} \preceq O_J, \boxdot)$, $(M! \preceq \texttt{TextN}, t)$, $(\texttt{TextN} \preceq O_J, t)$, $(M! \preceq \texttt{MobN}, m)$, $(\texttt{MobN} \preceq O_J, m)$, $(O_J \preceq \texttt{End}, \boxdot)$
- $\mathcal{C} := \{(R_1, \boxdot), (R_2, h), (R_3, \neg h), (R_4, a), (R_5, a), (R_6, \neg a), (R_7, \neg t \wedge \neg m), (R_7, t \wedge m)\}$, where
  - $R_1 := (\{\texttt{ProcR}, \texttt{CheckFR}\}, \{(\texttt{alice}, \texttt{bob}), (\texttt{alice}, \texttt{kate}), (\texttt{bob}, \texttt{kate})\})$
  - $R_2 := (\{\texttt{CheckFR}, \texttt{AntiML}\}, \{(\texttt{kate}, \texttt{mike})\})$
  - $R_3 := (\{\texttt{CheckFR}, \texttt{TaxFA}\}, \{(\texttt{bob}, \texttt{evie})\})$
  - $R_4 := (\{\texttt{CheckFR}, \texttt{PrepC}\}, \{(\texttt{bob}, \texttt{kate}), (\texttt{bob}, \texttt{ted}), (\texttt{kate}, \texttt{ted})\}$
  - $R_5 := (\{\texttt{PrepC}, \texttt{Sign}\}, \{(\texttt{kate}, \texttt{ted}), (\texttt{ted}, \texttt{kate})\})$
  - $R_6 := (\{\texttt{ProcR}, \texttt{Reject}\}, \{(\texttt{alice}, \texttt{alice}), (\texttt{bob}, \texttt{bob})\})$
  - $R_7 := (\{\texttt{End}\}, \emptyset)$

5 Checking Weak, Strong, and Dynamic Controllability

Before formally describing the three kinds of controllability, let us introduce them informally.

In general, *controllability* represents the capability by the Controller (i.e., the system managing the network) of assigning truth values to decision variables and values (i.e., resources) to variables, satisfying all the precedence and resource-related constraints and with any possible truth values of observations, which are not controllable. The different kinds of controllabilities differ with respect to how the Controller is able to deal with the (uncontrollable) observations.

Weak controllability refers to the capability of the Controller to guarantee the controllability when all the uncontrollable observations are known at the beginning, before assigning values to variables. Such kind of controllability is the weakest one, as observations are assumed to be known all before the Controller starts to manage the network.

Strong controllability, instead, refers to the capability of the Controller to guarantee controllability "in advance," without the need of taking into account the truth values of observations.

These two kinds of controllability represent a sort of lower and upper bound, respectively. The first one is weak, and often not acceptable in real-world contexts, as all observations are assumed to be known before the Controller works. On the other side, strong controllability is in turn rarely applicable as it assumes that the Controller is able to manage the network in a way independent from observations. But, in real-world contexts, as the one considered in the provided example, decisions are often intertwined with observations and are used to make the network flexible with respect to uncontrollable events. In this direction, the most interesting kind of controllability is the dynamic one.

Dynamic controllability refers to the capability of the Controller to use a dynamic strategy in assigning values to variables and truth values to decisions. The Controller is thus able to react to a truth assignment of an observation in a way that guarantees the controllability of the network with respect to every possible future value for observation. In other words, with respect to a given past assignment of resources and decisions and with respect to some known observations, the Controller is able to suitably manage every possible evolution of the network according to the currently unknown observations, guaranteeing the overall controllability.

As we have a network with both controllable features and uncontrollable ones, it is quite straightforward to formally specify the semantics of such controllabilities according to the game theory. In this case, we have two players. The first one is the Controller, able to set the controllable features, whereas the second one is Nature, deciding for the occurrences of the uncontrollable features. In the proposed game, Controller has to win with respect to any possible move of Nature. Different games represent different kinds of controllability.

To formally introduce the three main kinds of controllability, we start by defining a few crucial mappings that we make use of in the rest of this section. In the following definitions, we assume the existence of an underlying CCND $\mathcal{Z} = \langle \mathcal{X}, \mathcal{V}, D, \mathcal{O}, \mathcal{D}, \mathcal{P}, O, L, \preceq, \mathcal{C} \rangle$.

Definition 2 *((Un)controllable propositions)* The sets of controllable and uncontrollable propositions are defined by

$\mathcal{P}_\mathcal{D} := \{d \mid d \in \mathcal{P}, O(d) \in \mathcal{D}\}$ and $\mathcal{P}_\mathcal{O} := \{p \mid p \in \mathcal{P}, O(p) \in \mathcal{O}\}$, respectively.

In our example, $\mathcal{P}_\mathcal{D} := \{t, m\}$ and $\mathcal{P}_\mathcal{O} := \{h, a\}$.

Definition 3 *(Observation, decision, scenario)* An *observation* is a total mapping $\omega\colon \mathcal{P}_\mathcal{O} \mapsto \{0, 1\}$ assigning a truth value to *all uncontrollable* propositions. A *decision* is a total mapping $\delta\colon \mathcal{P}_\mathcal{D} \mapsto \{0, 1\}$ assigning a truth value to *all controllable* propositions. A scenario is a pair (ω, δ) merging an observation with a decision. A scenario s satisfies a label ℓ (in symbols, $\ell(s) = 1$) if ℓ is true under the interpretation given by s.

In our example, we have 4 possible observations and 4 possible decisions, namely,

	h	a
ω_1	0	0
ω_2	0	1
ω_3	1	0
ω_4	1	1

	t	m
δ_1	0	0
δ_2	0	1
δ_3	1	0
δ_4	1	1

Therefore, we have 16 possible scenarios.

Definition 4 *(Assignment, ordering)* An assignment is a total mapping $\alpha\colon \mathcal{X} \mapsto \mathcal{V}$ enforcing that $\alpha(X) \in D(X)$ for each $X \in \mathcal{X}$. An ordering (for $\mathcal{X}$) is a bijection $\pi\colon \mathcal{X} \rightleftharpoons \{1, \ldots, |\mathcal{X}|\}$.

What we introduced so far is enough to define traces.

Definition 5 *(Trace, constraint satisfaction)* A trace is a triple (s, α, π), where $s = (\omega, \delta)$ is a scenario, α an assignment, and π an ordering. An execution trace (s, α, π) satisfies the CCND $\mathcal{Z}$ (in symbols, $\mathcal{Z}(s, \alpha, \pi) = 1$) if and only if the following two conditions are met.

(1) For each $(X, Y, \ell) \in \preceq$, if $\ell(s) = 1$, then $\pi(X) < \pi(Y)$.
(2) For each $((S, T), \ell) \in \mathcal{C}$ with $S := \{X_j, \ldots, X_k\}$, if $\ell(s) = 1$, then $(\alpha(X_j), \ldots, \alpha(X_k)) \in T$.

We are now in position to define weak, strong, and dynamic controllability as two-player games. The first kind of controllability is *weak controllability* which assumes that the observation is known in advance.

Game 1 (Weak controllability) *The game takes place in two rounds. Nature plays first, then plays Controller.*

Round 1 *Nature chooses an observation ω.*
Round 2 *Controller chooses a decision δ, an assignment α, and an ordering π.*

Controller wins if $\mathcal{Z}((\omega, \delta), \alpha, \pi) = 1$. Nature wins otherwise.

Definition 6 *(Weak controllability)* A CCND is weakly controllable if Controller has a winning strategy for Game 1. It is weakly uncontrollable if it is Nature to have a winning strategy for it.

The CCND in Fig. 2a is weakly controllable. A winning strategy for controller is the following (we omit the symbols ω, δ, α, and π to ease reading). To give an example, suppose that Nature assigns false to both h and a. Then Controller, in this order, assigns `wfms` to `Start`, `alice` to `ProcR`, `wfms` P_S, `bob` to `CheckFR`, `wfms` to H?, `evie` to `TaxFA`, `wfms` to H_J, `wfms` to P_J, `wfms` to A?, `alice` to `Reject`, `wfms` to A_J, `wfms` to T!, false to t, `wfms` to M!, true to m, `server` to `EmailN`, `server` to `TextN`, `wfms` to O_J, and `wfms` to `End`. We discuss the other cases in Sect. 6 when we introduce ZETA.

The second kind of controllability is *strong controllability* which assumes that the observation will be revealed only when all variables have been executed (with respect to some order) and a decision has been chosen.

Game 2 (Strong controllability) *The game takes place in two rounds. Controller plays first, then plays Nature.*

Round 1 *Controller chooses a decision δ, an assignment α, and an ordering π.*
Round 2 *Nature chooses an observation ω.*

Controller wins if $\mathcal{Z}((\omega, \delta), \alpha, \pi) = 1$. Nature wins otherwise.

Definition 7 *(Strong controllability)* A CCND is strongly controllable if Controller has a winning strategy for Game 2. It is strongly uncontrollable if it is Nature to have a winning strategy for it.

The CCND in Fig. 2a is not strongly controllable. The problem lies in the constraints (R_2, h) and $(R_3, \neg h)$. Suppose that before starting Controller decides to assign `bob` to `CheckFR`. If during execution Nature assigns true to `hugeA?`, the process goes for `AntiML` with no user available for it since `bob` and `mike` are brothers. Then, Controller could change his mind and decide to assign `kate` (instead of `bob`) to `CheckFR` so that `mike` for `AntiML` would be fine as `kate` and `mike` are not relatives. But if during execution Nature assigns false to `hugeA?`, `evie` would not be fine for `TaxFA` as `evie` is `kate`'s sister. Therefore, Controller has no way to preassign a value to `CheckFR`.

Yet, the CCND is still executable as long as we decide (during execution) which value to assign to `CheckFR` *after* observing which truth value Nature has assigned to `hugeA`. This leads us to consider the most powerful kind of controllability: *dynamic controllability*.

Game 3 (Dynamic controllability) *Let* $\omega\colon \mathcal{P}_{\mathcal{O}} \mapsto \{0,1\}$, $\delta\colon \mathcal{P}_{\mathcal{D}} \mapsto \{0,1\}$, $\alpha\colon \mathcal{X} \mapsto \mathcal{V}$, *and* $\pi\colon \mathcal{X} \mapsto \mathbb{N}$ *be partial mappings, each undefined for all its domain elements. At the end of the game,* ω *is an observation,* δ *a decision,* α *an assignment, and* π *an ordering. The game proceeds in rounds until all variables and propositions have been assigned. Each round is as follows.*

1. *If all variables have been assigned, the game is over.*
2. *Otherwise, Controller chooses an unassigned variable* $X \in \mathcal{X}$, *a value* $v \in D(X)$, *and sets* $\alpha(X) := v$ *and* $\pi(X) := k$, *where* k *is the current number of assigned variables.*
   (a) *If* $X \in \mathcal{O}$, *let* p *be the uncontrollable proposition associated with* X. *Nature chooses a truth value* $b \in \{0,1\}$ *and sets* $\omega(p) := b$.
   (b) *If* $X \in \mathcal{D}$, *let* d *be the controllable proposition associated with* X. *Controller chooses a truth value* $b \in \{0,1\}$ *and sets* $\delta(p) := b$.

Controller wins if $\mathcal{Z}((\omega,\delta),\alpha,\pi) = 1$. *Nature wins otherwise.*

Definition 8 *(Dynamic controllability)* A CCND is dynamically controllable if Controller has a winning strategy for Game 3. Dynamically uncontrollable if it is Nature to have a winning strategy for it.

The CCND in Fig. 2a is dynamically controllable. Controller's strategy is the following (we only discuss the relevant assignments).

1. Controller assigns `wfms` to `Start`, `alice` to `ProcR`, `wfms` to P_S and `wfms` to H?, then
   (a) If Nature assigns false to h, then Controller assigns `bob` to `CheckFR`, `evie` to `TaxFA`, `wfms` to H_J, `wfms` to P_J and `wfms` to A?. Then,
      i. If Nature assigns false to a, then Controller assigns `alice` to `Reject`, `wfms` to A_J, `wfms` to T!, false to t, `wfms` to M!, true to m, `server` to `EmailN`, `server` to `MobN`, `wfms` to O_J, and `wfms` to `End`.
      ii. If Nature assigns true to a, Controller assigns `kate` to `PrepC`, `ted` to `Sign`, `wfms` to A_J, `wfms` to T!, false to t, `wfms` to M, true to m, `server` to `EmailN`, `server` to `MobN`, `wfms` to O_J, and `wfms` to `End`.
   (b) If Nature assigns true to h, then Controller assigns `kate` to `CheckFR`, `mike` to `AntiML`, `wfms` to H_J, `wfms` to P_J, and `wfms` to A?. Then,
      i If Nature assigns false to a, then Controller assigns `alice` to `Reject`, `wfms` to A_J, `wfms` to T!, false to t, `wfms` to M!, true to m, `server` to `EmailN`, `server` to `MobN`, `wfms` to O_J, and `wfms` to `End`.
      ii If Nature assigns true to a, then Controller assigns `ted` to `PrepC`, `kate` to `Sign`, `wfms` to A_J, `wfms` to T!, false to t, `wfms` to M!, true to m, `server` to `EmailN`, `server` to `MobN`, `wfms` to O_J, and `wfms` to `End`.

Overall, the important thing is to execute H? *before* executing `CheckFR` (since, being in a parallel block, `CheckFR` could be executed before H?).

Corollary 1 (Implication chains) strong controllability $\Rightarrow$ dynamic controllability $\Rightarrow$ weak controllability.

Corollary 1 holds by definition (more specifically, see Definitions 6, 7, and 8).

We conclude this section by classifying the computational complexity of deciding weak, strong, and dynamic controllability of CCNDs. The complexity of deciding weak, strong, and dynamic controllability of CNCUs was investigated in [55]. As a result, since CNCUs are a special case of CCNDs we immediately inherit hardness results. Therefore, what we are really left to prove in order to prove completeness is membership.

Theorem 1 *Deciding weak controllability of CCNDs is* Π_2^p*-complete.*

Proof **Hardness:** Inherited from CNCUs. **Membership:** For each observation ω, there exists a decision δ, an assignment α and an ordering π such that $\mathcal{Z}(s,\delta,\pi) = 1$ where $s := (\omega,\delta)$. Since all these parts have a polynomial length, weak controllability is in Π_2^p by definition. □

Theorem 2 *Deciding strong controllability of CCNDs is NP-complete.*

Proof **Hardness:** Inherited from CNCUs. **Membership:** A certificate of yes is a decision δ plus an assignment α plus an ordering π. It is clear that the overall sum of these components has polynomial length since each of them has. To verify that $\mathcal{Z}((\omega,\delta),\delta,\pi) = 1$ for each possible ω, we just check satisfaction of all partial order and relational constraints for which δ satisfies the part of the labels involving controllable propositions only. Since we have a finite number of constraints we know that this check runs in polynomial time.

Theorem 3 *Deciding dynamic controllability of CCNDs is PSPACE-complete.*

Proof **Hardness:** Inherited from CNCUs. **Membership:** Algorithm 1 is a polynomial space algorithm to decide dynamic controllability of any CCND. It explores an AND/OR search tree whose depth size is upper bounded by $\mathcal{O}(|\mathcal{X}|)$[2]

[2] This is the only exception in the paper in which we overload symbols. Here, "$\mathcal{O}$" means "big-O" (growth of functions).

Algorithm 1: `ccnd-dc`$(\mathcal{Z})$

Input: A CCND $\mathcal{Z} = \langle \mathcal{X}, \mathcal{V}, D, \mathcal{O}, \mathcal{D}, \mathcal{P}, O, L, \preceq, \mathcal{C} \rangle$
Output: `Yes`, if $\mathcal{Z}$ is dynamically controllable. `No` otherwise.

ccnd-dc $(\mathcal{Z})$
  Let $\omega : \mathcal{P}_{\mathcal{O}} \mapsto \{0, 1\}$ be a partial mapping undefined for all of its domain elements
  Let $\delta : \mathcal{P}_{\mathcal{D}} \mapsto \{0, 1\}$ be a partial mapping undefined for all of its domain elements
  Let $\alpha : \mathcal{X} \mapsto \mathcal{V}$ be a partial mapping undefined for all of its domain elements
  Let $\pi : \mathcal{X} \mapsto \mathbb{N}$ be a partial mapping undefined for all of its domain elements
  return `Variable`$(\mathcal{Z}, \omega, \delta, \alpha, \pi, \emptyset)$

`Variable` $(\mathcal{Z}, \omega, \delta, \alpha, \pi, \overline{\mathcal{X}})$ ▷ $\overline{\mathcal{X}}$ keeps track of executed vars
  if $\overline{\mathcal{X}} = \mathcal{X}$ **then**
    return $\mathcal{Z}((\omega, \delta), \alpha, \pi)$
  for $X \in \mathcal{X} \setminus \overline{\mathcal{X}}$ **do** ▷ for each unexecuted var
    if `Value`$(\mathcal{Z}, \omega, \delta, \alpha, \pi, \overline{\mathcal{X}}, X)$ **then return** `Yes`;
  return `No`

`Value` $(\mathcal{Z}, \omega, \delta, \alpha, \pi, \overline{\mathcal{X}}, X)$
  for $v \in D(X)$ **do** ▷ for each element in X's domain
    Let α' be the extension of α in which $\alpha'(X) := v$
    Let π' be the extension of π in which $\pi'(X) := |\overline{\mathcal{X}}| + 1$
    if $X \notin \mathcal{D} \cup \mathcal{O}$ **then** ▷ Case 1
      if `Variable`$(\mathcal{Z}, \omega, \delta, \alpha', \pi', \overline{\mathcal{X}} \cup \{X\})$ **then**
        return `Yes`
    if $X \in \mathcal{O}$ **then** ▷ Case 2
      Let p be the proposition associated with X
      if `Observation`$(\mathcal{Z}, \omega, \delta, \alpha', \pi', \overline{\mathcal{X}} \cup \{X\}, p)$ **then return** `Yes`;
    if $X \in \mathcal{D}$ **then** ▷ Case 3
      Let d be the proposition associated with X
      if `Decision`$(\mathcal{Z}, \omega, \delta, \alpha', \pi', \overline{\mathcal{X}} \cup \{X\}, d)$ **then return** `Yes`;
  return `No`

`Observation` $(\mathcal{Z}, \omega, \delta, \alpha, \pi, \overline{\mathcal{X}}, p)$
  Let ω' be the extension of δ in which $\omega'(d) := 0$
  Let ω'' be the extension of δ in which $\omega''(d) := 1$
  return `Variable`$(\mathcal{Z}, \omega', \delta, \alpha, \pi, \overline{\mathcal{X}}) \wedge$ `Variable`$(\mathcal{Z}, \omega'', \delta, \alpha, \pi, \overline{\mathcal{X}})$

`Decision` $(\mathcal{Z}, \omega, \delta, \alpha, \pi, \overline{\mathcal{X}})$
  Let δ' be the extension of δ in which $\delta'(d) := 0$
  Let δ'' be the extension of δ in which $\delta''(d) := 1$
  return `Variable`$(\mathcal{Z}, \omega, \delta', \alpha, \pi, \overline{\mathcal{X}}) \vee$ `Variable`$(\mathcal{Z}, \omega, \delta'', \alpha, \pi, \overline{\mathcal{X}})$

Finally, it is easy to see that when a CCND contains no uncertainty (i.e., when $\mathcal{O} = \emptyset$) we deal with a consistency problem whose corresponding decision version is NP-complete. Indeed, NP-hardness is inherited from CNs which are a special case when $\mathcal{D}$ and $\preceq$ are also empty. Membership in NP boils down to verify a certificate made up of a decision δ, an assignment α and an ordering π the same way discussed in the proof of Theorem 2.

These complexity results suggest immediately how to implement rough strategy synthesis algorithms. To avoid heaving the paper, we only discuss what they look like.

For dynamic controllability, we need to modify Algorithm 1 for it to save the strategy tree instead of only traversing it. Specifically, whenever Algorithm 1 would return "yes," we return a (sub)tree containing the part of δ, α and π that is relevant at that specific recursion level. Whenever Algorithm 1 would return "no," we return an empty tree. If the CCND is not dynamically controllable, then we get an empty tree.

For weak controllability, we proceed this way. We iterate on each observation ω and generate a CND (i.e., a CCND with $\mathcal{O} = \emptyset$) in which we fix the values of all uncontrollable propositions according to ω. After that, we use Algorithm 1 to synthesize α, δ, π for this specific ω (note that no AND nodes are explored here). If just one of these trees is empty, the CCND is not weakly controllable.

For strong controllability, we generate a CND by wiping out all uncontrollable propositions from the initial CCND (wherever they appear). After that, we run Algorithm 1 to get α, δ, π (if any) for all ω (again, no AND nodes are explored). If we get an empty tree, the CCND is not strongly controllable.

6 Zeta: a tool for CCNDs

The initial version of ZETA was a Java tool provided in [59,61] that acted as a strategy synthesizer and executor simulator for weakly, strongly and dynamically controllable CNCUs according to the old approach. Taking advantage of the new results in this paper, we completely rewrote ZETA in C++ in order to implement the strategy synthesis algorithms we discussed at the end of Sect. 3. In its internal, ZETA exploits pruning techniques that stop a search when consistency of a partial trace $((\omega, \delta), \alpha, \pi)$ results broken already in the search tree and also optimization techniques aimed at ignoring irrelevant variables. Despite ZETA deals with weak and strong controllability as well, it was mainly conceived to address dynamic controllability of CCNDs natively, with special attention devoted to the synthesis of dynamic strategies and execution of dynamically controllable networks. ZETA relies on Z3's C++ API [35] as a backend for solving CNDs which play a role when synthesizing strategies for weakly and strongly controllable CCNDs. Instead, to synthesize strategy trees for dynamically controllable CCNDs ZETA does not need Z3.

We added new features and simplified the input language. We discuss this new version in the rest of this section, we compare with the experimental evaluation in [61] and we also provide a new set of benchmarks for CCNDs.

6.1 The New Version of Zeta and Its Experimental Evaluation

Listing 1 shows ZETA's help screen. Despite the command line arguments follow a different BNF grammar, this version of ZETA also offers the possibility to:

- decide dynamic controllability of a CCND without synthesizing any strategy (i.e., decision problem),
- print a previously synthesized strategy in a human readable format,

Listing 1 ZETA's help screen.

```
$ ./zeta network ACTION [-silent]

ACTION ::= -d CONTROLLABILITY          # decision problems
         | -s CONTROLLABILITY strategy # strategy synthesis
         | -x[i] strategy [N]          # execution (online planning and scheduling)
         | -p strategy                 # strategy printing (human readable)

CONTROLLABILITY ::= weak | strong | dynamic
```

– execute a controllable CCND interactively meaning that the user is free to play the Nature (i.e., choose the uncontrollable truth value assignments upon the execution of the observation variables).

The new input language of ZETA consists of three main sections instead of five. Listing 2 shows the specification of the CCND in Fig. 2a written in ZETA's input language, where:

– the section `Variables` specifies the sets $\mathcal{X}$, $\mathcal{P}$, $\mathcal{V}$ as well as the mappings O and L,
– the section `Precedence` specifies $\preceq$, and
– the section `Constraints` specifies $\mathcal{C}$.

We added a few comments to clarify the syntax.

If the input network is controllable (or consistent) and ZETA is asked to synthesize a strategy for a specific kind of controllability, ZETA saves to file the strategy tree if the network is proved consistent or controllable. Such a strategy will be provided later as input (along with the network) in the execution phase.

We run ZETA on the specification in Listing 2 to prove that the CCND in Fig. 2 is weakly, dynamically but not strongly controllable (Listing 3) and we carried out 1000 execution simulations for weak and dynamic controllability generating random observations at each simulation (we show a few in Listings 4 and 5). We used a FreeBSD virtual machine run on top of a VMWare ESXi Hypervisor using a physical machine equipped with an Intel i7 2.80 GHz and 16 GB of RAM. The VM was assigned 12 GB of RAM and full CPU power. Time and space consumed are negligible for the CCND in Fig. 2 apart from the synthesis of the strategy tree for dynamic controllability that took about 4.6 seconds.

In [59] and [61] two different sets of benchmarks for CNCUs were provided. However, in [59] no particular criteria was adopted to generate the CNCUs. Instead, in [61] CNCUs were divided with respect to some criteria of interest.

We compared the new approach against the one in [61] and converted in the new format 100 weakly controllable CNCUs only, 100 strongly (therefore weakly and dynamically) controllable CNCUs, 100 dynamically but not strongly controllable CNCUs and finally 100 CNCUs uncontrollable for each kind of controllability as the reverse implication chain *weak uncontrollability* $\Rightarrow$ *dynamic uncontrollability* $\Rightarrow$ *strong uncontrollability* holds too. We put aside strongly and dynamically uncontrollable CNCUs in [61] as they could be controllable even in the old approach for some other kind of controllability (that is why we focused on weakly uncontrollable CNCUs only).

Each CNCU has exactly 6 variables, where each variable has the same 6 values in its domain and specifies a maximum number of relational constraints of 40% of $|\mathcal{X}| \times |\mathcal{O}|$, where each binary relation $((\{X_i, X_j\}, T), \ell)$ has a maximum number of tuples of 50% of $|D(X_i)| \times |D(X_j)|$ and the label ℓ is generated randomly. Furthermore, to increase the search space all variables are *unlabeled* and no partial order is specified.

Figure 3 shows the comparison between the old approach (dashed lines) and the new one (solid lines) where the captions of the subfigures say which set of benchmarks is under analysis. The data shows that the approach in this paper is faster. The new ZETA also proved that `weak/6vars/6obs/008` (classified as weakly controllable only by the old approach) is also dynamically controllable when considering dynamic orderings.

After that, we generated a new set of benchmarks for CCNDs. We followed the previous idea of generating CCNDs that are weakly controllable only, strongly controllable, dynamically but not strongly controllable and weakly uncontrollable. For each of these subsets, we generated 100 CCNDs having 1 decision and 1 observation variables, 100 CCNDs having 2 decision and 2 observation variables, 100 CCNDs having 3 decision and 3 observation variables and 100 CCNDs having 4 decision and 4 observation variables. That is, in each network the number of decision variables is always equal to the number of observation variables. In this way half of the propositions are controllable and the other half uncontrollable. Each CCND has exactly 8 unlabeled variables whose domains are filled randomly from a set of 8 values. Let N be the number of decision and observation variables in a CCND. The partial order relation is filled with maximum $2 + N$ randomly labeled order edges, whereas the constraint set is filled with maximum $1 + N$ randomly labeled

Listing 2 Specification of Fig. 2 in ZETA's input language.

```
Variables {                        # Syntax
    Start : wfms : ;               # variable : domain : label;
    ProcR : alice bob : ;
    PS : wfms : ;
    CheckFR : bob kate : ;
    H? : h : wfms : ;              # observation_variable : proposition : domain : label;
    AntiML : mike : h;
    TaxFA : evie : !h;
    HJ : wfms : ;
    PJ : wfms : ;
    A? : a : wfms : ;
    PrepC : kate ted : a;
    Sign : kate ted : a;
    Reject : alice bob : !a;
    AJ : wfms : ;
    T! : t : wfms : ;              # decision_variable : proposition : domain : label;
    M! : m : wfms : ;
    EmailN : server : ;
    TextN : server : t;
    MobN : server : m;
    OJ : wfms : ;
    End : wfms : ;
}

Precedence {                       # variable -> variable : label;
    Start -> ProcR : ;
    ProcR -> PS: ;
    PS -> CheckFR : ;
    CheckFR -> PJ : ;
    PS -> H : ;
    H -> AntiML : h;
    AntiML -> HJ : h;
    H -> TaxFA : !h;
    TaxFA -> HJ : !h;
    HJ -> PJ : ;
    PJ -> A : ;
    A -> PrepC : a;
    PrepC -> Sign : a;
    Sign -> AJ : a;
    A -> Reject : !a;
    Reject -> AJ : !a;
    AJ -> T : ;
    T -> M : ;
    M -> EmailN : ;
    EmailN -> OJ : ;
    M -> TextN : t;
    TextN -> OJ : t;
    M -> MobN : m;
    MobN -> OJ : m;
    OJ -> End : ;
}

Constraints {                      # scope : tuples : label;
    ProcR CheckFR : (alice bob) (alice kate) (bob kate) : ;
    CheckFR PrepC : (bob kate) (bob ted) (kate ted) : a;
    PrepC Sign : (kate ted) (ted kate) : a;
    CheckFR AntiML : (kate mike) : h;
    CheckFR TaxFA : (bob evie) : !h;
    ProcR Reject : (alice alice) (bob bob) : !a;
    End : : !t !m;
    End : : t m;
}
```

binary relations (each containing maximum $2 + N$ randomly generated tuples). Figure 4 shows the analysis carried out with ZETA where, again, the captions of the subfigures say which set of benchmarks is under analysis. Also, note that this time the size of the strategy is given in number of nodes.

Listing 3 Weak, strong and dynamic strategy synthesis for Fig. 2.

```
$ ./zeta LOP_jods.ccnd -s weak LOP_jods.weak
    .s
controllable
$ ./zeta LOP_jods.ccnd -s strong LOP_jods.
    strong.s
uncontrollable
$ ./zeta LOP_jods.ccnd -s dynamic LOP_jods.
    dynamic.s
controllable
```

We executed all CNCUs and all CCNDs 1000 times each with respect to weak, strong, and dynamic controllability (according to the set of benchmarks under analysis). No execution crashed.

The new version of ZETA (for FreeBSD, Linux, and Windows) along with the running example of this paper and the experimental evaluation we just discussed is available at https://github.com/matteozavatteri/zeta inside the directory `ccnd`.

6.2 Limitations

Limitations in the considered experimental setting are mainly due to the fact that ZETA has been realized as a sound proof-of-concept prototype for checking controllabilities of CCNDs. We decided to generate artificial networks to have a generic set of networks with predefined features, leaving for the future work an experimental evaluation on real-world networks, possibly derived from business processes. Having real-world business processes for extensive benchmarking and comparison of different approaches is in fact still an issue in business process research [13] and some initiatives have been proposed to this end [13,48]. Indeed, on one hand, business processes are still considered an important piece of strategic knowledge, preventing their sharing even for research purposes. On the other hand, such processes would need to be suitably mapped into new networks, according to the proposed theoretical framework. Such a step may present different kinds of issues, as some features of the processes could be either not supported (e.g., metric constraints between tasks) or underspecified (e.g., the resources available for specific tasks). To partially deal with the real-world applicability of our approach, we discussed a concrete (but simplified with respect to the other features that we did not consider) example throughout the paper.

From a more technical point of view, the limitations of ZETA, once a process is encoded into a CCND (a linear time step), are related to its main algorithmic bottlenecks. Indeed, a combinatorial explosion may arise, mainly due to the following aspects:

- the number of (variables modeling) activities in the process,
- the number of (values modeling) resources associated with activities in the process,
- the number of (Booleans modeling) XOR conditions/decisions in the process, and
- the number of possible total orders arising from parallel blocks in the process.

These aspects can be considered in isolation or simultaneously, in combination. However, in general, we can say that *the more, the worse*, as is clear also from the theoretical complexity analysis that we carried out at the end of Sect. 5.

7 Related Work

A *constraint network (CN,* [14]) consists of a finite set of variables, a set of finite discrete domains (one for each variable), and a set of relational constraints, and is a possible formalism to model the constraint satisfaction problem (CSP) that is the problem of finding an assignment of values to the variables satisfying all constraints. Deciding consistency of CNs is NP-complete [14]. Over the years, several algorithms for computing one or all solutions of a CN were provided following search or inference approaches (or a combination of both). For example, node, arc, and path consistency [29,32] are incomplete filtering techniques aimed to rule out inconsistent assignments. A CN is minimal if any tuple of any relation can be extended to a complete consistent solution [14,33]. Computing a minimal network is NP-hard [33]. Moreover, given a minimal network, generating an arbitrary solution is NP-hard as well [26]. Directional consistency was introduced to speed up the process of consistency checking and solution generation exploiting total orderings on the set of variables [17]. On top of that, when we need to generate a solution to a CN limiting or avoiding backtracking, we need, in general, a concept of (strong) k-consistency which guarantees that any consistent assignment to $k-1$ variables can be extended to k variables without breaking consistency [24,25]. CNs model fully controllable CSPs only. This work deals with a CSP in which some parts are uncontrollable.

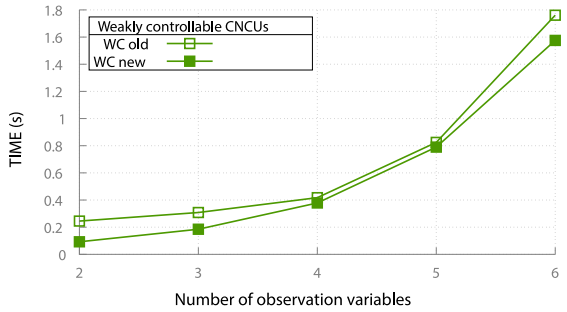
When we need dynamicity (i.e., CSPs changing the set of variables and/or constraints) over time, we need to consider *dynamic CSPs*. Dechter and Dechter in [15] defined a dynamic CSP as a sequence of static CSPs where each CSP in the sequence differs from the previous one for a single change in the constraint or in the variable set and provided algorithms for support propagation and contradiction resolution for acyclic CNs. The degree of support of each variable

Listing 4 Weak execution simulations for the CCND in Fig. 2.

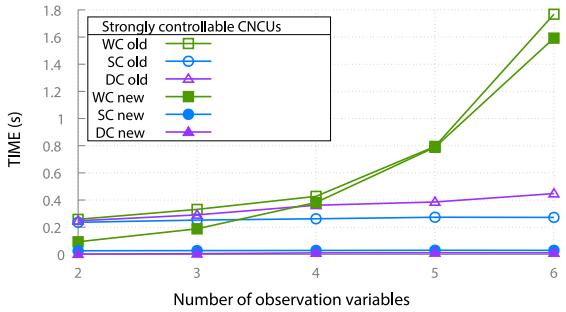
```
$ ./zeta LOP_jods.ccnd -x LOP_jods.weak.s 1000
-------------    -------------    -------------    -------------
Execution 1      Execution 2      Execution 3      Execution 4
-------------    -------------    -------------    -------------
h=0              h=0              h=1              h=1
a=0              a=1              a=0              a=1
Start=wfms       Start=wfms       Start=wfms       Start=wfms
ProcR=alice      ProcR=alice      ProcR=alice      ProcR=alice
PS=wfms          PS=wfms          PS=wfms          PS=wfms
CheckFR=bob      CheckFR=bob      CheckFR=kate     CheckFR=kate
H=wfms           H=wfms           H=wfms           H=wfms
TaxFA=evie       TaxFA=evie       AntiML=mike      AntiML=mike
HJ=wfms          HJ=wfms          HJ=wfms          HJ=wfms
PJ=wfms          PJ=wfms          PJ=wfms          PJ=wfms
A=wfms           A=wfms           A=wfms           A=wfms
Reject=alice     PrepC=ted        Reject=alice     PrepC=ted
AJ=wfms          Sign=kate        AJ=wfms          Sign=kate
T=wfms, t=1      AJ=wfms          T=wfms, t=1      AJ=wfms
M=wfms, m=0      T=wfms, t=1      M=wfms, m=0      T=wfms, t=1
EmailN=server    M=wfms, m=0      EmailN=server    M=wfms, m=0
TextN=server     EmailN=server    TextN=server     EmailN=server
OJ=wfms          TextN=server     OJ=wfms          TextN=server
End=wfms         OJ=wfms          End=wfms         OJ=wfms
                 End=wfms                          End=wfms
-------------    -------------    -------------    -------------
SAT!             SAT!             SAT!             SAT!
-------------    -------------    -------------    -------------
```

Listing 5 Dynamic execution simulations for the CCND in Fig. 2.

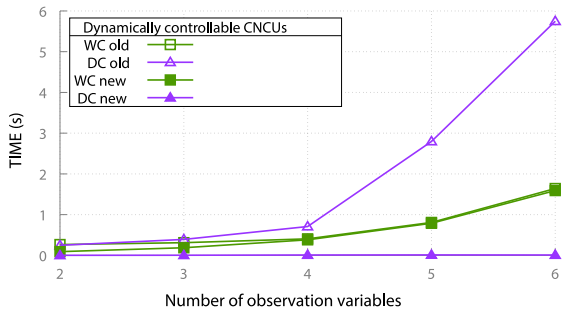
```
$ ./zeta LOP_jods.ccnd -x LOP_jods.dynamic.s 1000
-------------    -------------    -------------    -------------
Execution 1      Execution 2      Execution 3      Execution 4
-------------    -------------    -------------    -------------
Start=wfms       Start=wfms       Start=wfms       Start=wfms
ProcR=alice      ProcR=alice      ProcR=alice      ProcR=alice
PS=wfms          PS=wfms          PS=wfms          PS=wfms
H=wfms, h=0      H=wfms, h=0      H=wfms, h=1      H=wfms, h=1
CheckFR=bob      CheckFR=bob      CheckFR=kate     CheckFR=kate
TaxFA=evie       TaxFA=evie       AntiML=mike      AntiML=mike
HJ=wfms          HJ=wfms          HJ=wfms          HJ=wfms
PJ=wfms          PJ=wfms          PJ=wfms          PJ=wfms
A=wfms, a=0      A=wfms, a=1      A=wfms, a=0      A=wfms, a=1
Reject=alice     PrepC=kate       Reject=alice     PrepC=ted
AJ=wfms          Sign=ted         AJ=wfms          Sign=kate
T=wfms, t=0      AJ=wfms          T=wfms, t=0      AJ=wfms
M=wfms, m=1      T=wfms, t=0      M=wfms, m=1      T=wfms, t=0
EmailN=server    M=wfms, m=1      EmailN=server    M=wfms, m=1
MobN=server      EmailN=server    MobN=server      EmailN=server
OJ=wfms          MobN=server      OJ=wfms          MobN=server
End=wfms         OJ=wfms          End=wfms         OJ=wfms
                 End=wfms                          End=wfms
-------------    -------------    -------------    -------------
SAT!             SAT!             SAT!             SAT!
-------------    -------------    -------------    -------------
```



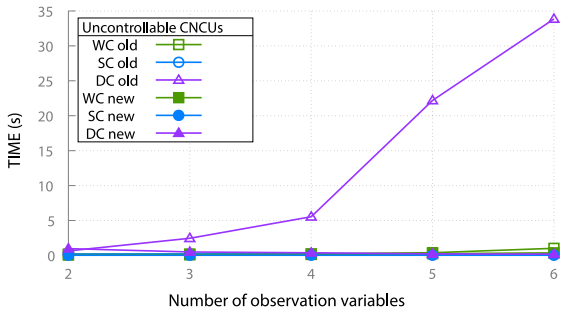
(a) `AAI/weak/6vars/*obs/`



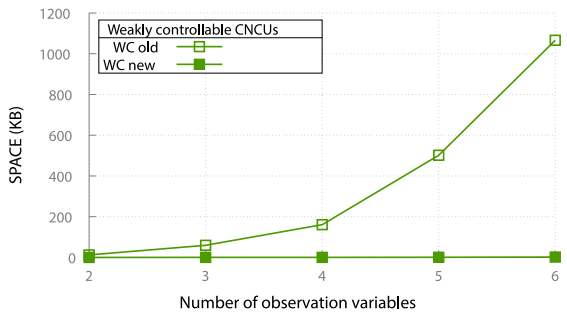
(b) `AAI/strong/6vars/*obs/`



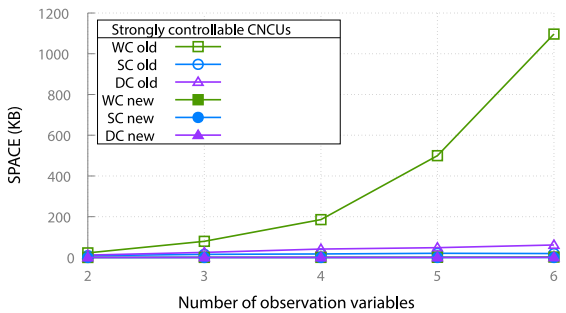
(c) `AAI/dynamic/6vars/*obs/`



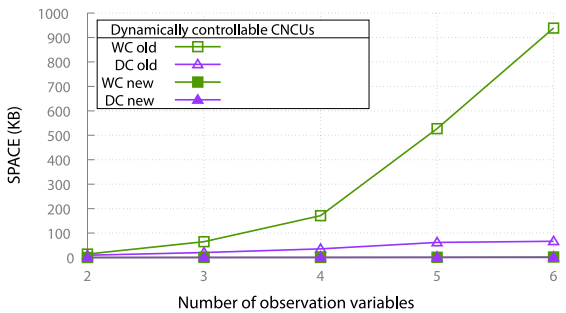
(d) `AAI/uncontrollable/6vars/*obs/`



(e) `AAI/weak/6vars/*obs/`

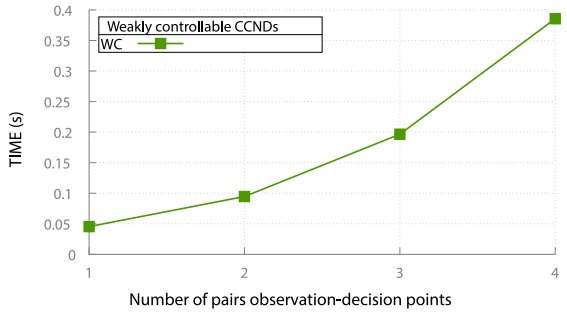


(f) `AAI/strong/6vars/*obs/`

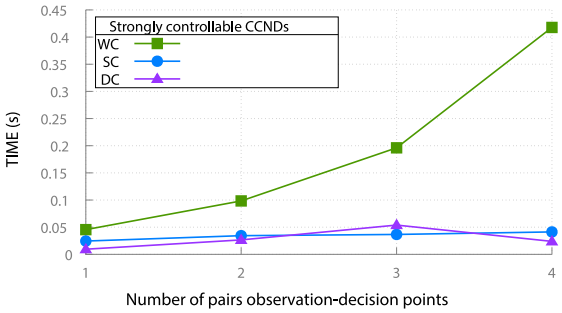


(g) `AAI/dynamic/6vars/*obs/`

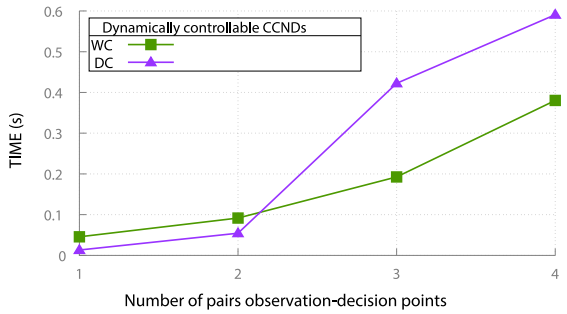
Fig. 3 Time and space comparison between the new approach and the old one in [61]. The sizes of the strategies in the new approach (number of nodes) have been converted in kilobytes according to the data structure used to save them to file



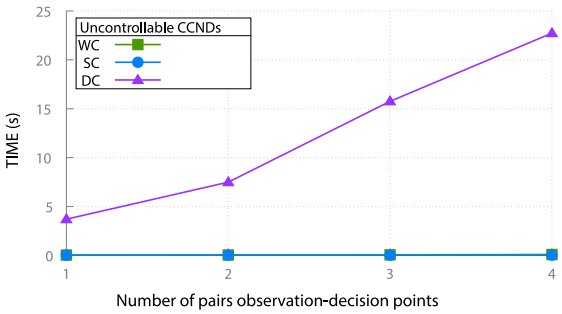
(a) `ccnd/weak/8vars/*o*d/`



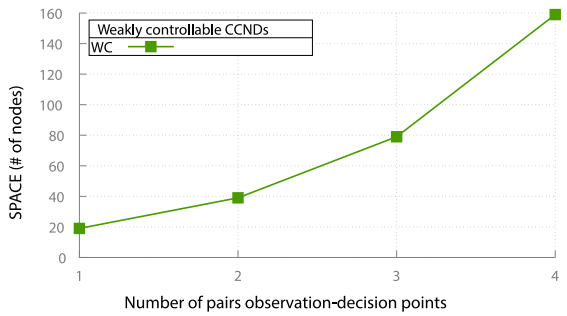
(b) `ccnd/strong/8vars/*o*d/`



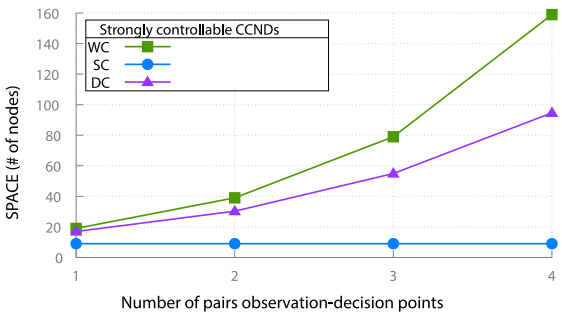
(c) `ccnd/dynamic/8vars/*o*d/`



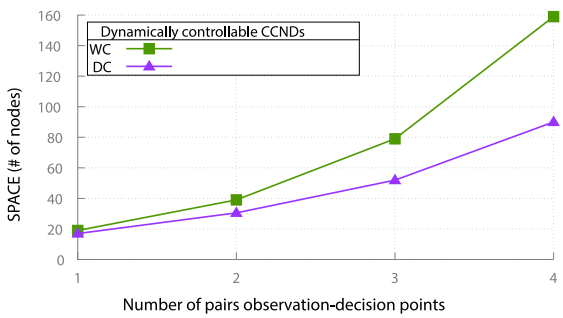
(d) `ccnd/uncontrollable/8vars/*o*d/`



(e) `ccnd/weak/8vars/*o*d/`



(f) `ccnd/strong/8vars/*o*d/`



(g) `ccnd/dynamic/8vars/*o*d/`

Fig. 4 Time and space analysis on the new set of benchmarks (CCNDs)

models the number of consistent extensions that any variable assignment has with respect to the set of all solutions. No notion of controllability is given.

Despite the terminology is similar, another approach to tackle *dynamic constraint satisfaction problems (DCSPs)* was provided in [31] by Mittal and Falkenhainer who extended a classic CSP by partitioning the set of constraints in *compatibility constraints* (i.e., classic CSP constraints) and *activity constraints*, a new kind of constraints saying when variables are *active* (i.e., when they must belong to a solution) depending on which values other variables are assigned. Solving a DCSP means finding all (complete) solutions satisfying all compatibility and activity constraints. DCSPs can model *configuration problems*. Amilhastre et al. investigated in [1] the interactive version of a DCSP via *Assumption-based CSPs (A-CSPs)* and encoded the problem into an (exponential-size) automaton in order to pre-compute the set of possible solutions to allow for fast generation of a solution. Sabin and Freuder proposed in [40] to use the term *conditional CSP (CCSP)* instead of DCSP in order to make more clear that configuration problems are conditional problems. However, these types of DCSPs (A-CSPs included) are *controllable conditional CSPs* (that is why consistency analysis is enough).

To face uncertainty, a line of research has been proposed over the years providing several formalisms built on top of (temporal) constraint networks.

In [20], Fargier and Lang provided a probabilistic approach to a CSP under uncertainty where each relational constraint is associated with a probability of being part of the real problem and provided an algorithm to compute the most probable solutions (those that are expected to satisfy the constraints of the real problem). Later, in [21], the same authors along with Martin-Clouaire and Schiex defined a *probabilistic CSP (PCSP)* partitioning the set of variables in *parameters* (i.e., variables whose value assignments are out of control) and *decision variables* (variables whose value assignments are under control). Constraints are specified over both parameters and decision variables. Each parameter is also associated with a probability. The authors provided two algorithms to find solutions when (1) no knowledge on the uncontrollable part arrives "in time" before assigning decision variables, and (2) all knowledge on the uncontrollable parts reveals before assigning decision variables. However, no approach to deal with partial observation is investigated in order to assign decision variables relying on the knowledge of the uncontrollable part revealed "so far." In this paper, we have a notion of partial observation when we deal with dynamic controllability.

In [22], Fargier et al. defined a *mixed CSP* where the set of variables is still divided in parameters and decision variables but no probability measure to constraints is given. In that work, the authors focused on the problem of consistency under *full observability* and *no observability* of the uncontrollable part. However, consistency is not investigated under partial observability (i.e., no dynamic context).

A constraint network under conditional uncertainty (CNCU) [59,61] extends a CN with a set of *Boolean propositions* whose truth value assignments *are out of control* (or, equivalently, can be thought of as being under the control of Nature), *observation variables* (i.e., special kind of variables to observe such truth value assignments), and *propositional labels* to enable or disable a subset of variables and constraints. CNCUs also introduce an (implicit) notion of *partial order* among the variables. In [59,61], the authors extended algorithms based on directional consistency [17] to accommodate labels in order to synthesize execution strategies for weakly, strongly, and dynamically controllable CNCUs with respect to total orders. More recently, an algorithm to decide dynamic controllability of a CNCU considering dynamic orderings was provided in [55]. Regarding computational complexity aspects of CNCUs, weak controllability is Π_2^p-complete, strong controllability is NP-complete, whereas dynamic controllability is PSPACE-complete [55].

CNCUs were strongly inspired from research on controllability of temporal networks. A *simple temporal network* (*STN*, [16]) is a specialization of a CN where variables are called time points, have continuous domain, and model the occurrence of some temporal events as soon as they are assigned real values, whereas constraints are linear inequalities limiting the minimal and maximal temporal distance between pairs of time points. STNs model fully controllable temporal plans only, therefore consistency analysis (which is in P) is enough. *Simple Temporal Networks with Uncertainty* (*STNUs*, [34,44]) add uncontrollable (but bounded) durations between pairs of temporal events (temporal uncertainty), whereas *conditional simple temporal networks* (*CSTNs*, [28,43]) extend STNs by labeling time points and constraints with conjunctions of literals whose truth value assignments are out of control (conditional uncertainty). *Conditional simple temporal networks with uncertainty* (*CSTNUs*, [27]) merge STNUs and CSTNs, whereas *conditional simple temporal networks with uncertainty and decisions* (*CSTNUDs*, [49,60]) encompass all previous formalisms by adding further Boolean propositions whose truth value assignments are under control. Weak controllability of CSTNUDs remains unexplored, whereas strong controllability is addressed in [56] and dynamic controllability in [49,60].

Some proposals also extended STNUs and CSTNUs in order to deal with resources and temporal aspects simultaneously. In [12], a workflow and a fragment of *Temporal Role-Based Access Control* (*TRBAC*, [2]) are encoded into an STNU, and security policies are modeled by security constraints (SCs) along with security constraint propagation rules (SCPRs) that propagate them depending on which user is executing which time point. Dynamic controllabil-

ity checking for this augmented network is addressed in [11], which revised the work in [12] with the proposal of *conditional simple temporal networks with uncertainty and resources (CSTNURs)*. *Access controlled temporal networks (ACTNs)* were proposed in [10] again to handle time and resources simultaneously and differ from CSTNURs for the type of employed constraints and blocking conditions on resources. ACTNs and CSTNURs do not employ controllable conditional constraints. This work does not address temporal constraints (in a quantitative sense) but allows for handling all possible users assignments (with respect to the total order of the components) during execution whereas CSTNURs and ACTNs don't.

The problem of verifying WF features related to the assignment of tasks to users is known in the literature as WF satisfiability and resiliency [45]. The *workflow satisfiability problem (WSP)* is the problem of finding an assignment of tasks to users (i.e., a plan) such that the execution of the WF gets to the end satisfying all authorization constraints. The *workflow resiliency problem* is the WSP under the uncertainty that a maximum number of users may become (temporally) absent before or during execution (see [62] for a recent controller synthesis approach). In this work, we dealt with a *dynamic* WSP encoding an ACWF into a CCND for both checking controllability and executing the WF.

In the context of business process management (BPM), Cabanillas et al. investigated resource allocation for business processes in [6]. They consider an RBAC environment and they do not impose any particular order on activities. They also address loops but their approach does not address *History-Based Allocation* of resources. This work addresses history-based allocation of resources exploiting results from dynamic controllability of CCNDs.

Zavatteri et al. proposed in [51] an initial approach to check weak, strong, and dynamic controllability of access controlled workflows under conditional uncertainty. Such workflows involve a set of partially ordered tasks (variables of the CSP), authorized users (domains), and authorization constraints (relational constraints). Conditional uncertainty models uncontrollable XOR splits in the workflow meaning that once executed we cannot decide in which workflow path the execution will continue. In [51], the authors unfold workflow paths and map them to classic CNs to reason on the intersection of common parts when dealing with a dynamic user assignment (different users assigned to the same tasks depending on which workflow path is being taken). The proposed approach pointed out that dynamic controllability might be a matter of how the components of the workflow are ordered, a hypothesis that was later confirmed by Zavatteri and Viganò with the proposal of *constraint networks under conditional uncertainty (CNCUs)*, a formalism suitable to model resource allocation under conditional uncertainty in modern business processes. After that, the work in [54] provided an initial encoding from ACWFs to CNCUs to exploit existing algorithms and software in order to handle such an issue automatically. This paper extends that work.

Further recent research addressed the computational complexity of several kinds of controllability of processes with also uncontrollable resource assignments and uncontrollable availability of resources [57,58]. See also [50] for a summary of temporal and resource controllability in the BPM context employing a constraint-based approach.

8 Conclusions and Future Work

We defined conditional constraint networks with decisions (CCNDs) by extending constraint networks under conditional uncertainty (CNCUs) for them to support:

1. decision variables, and
2. a labeled partial order relation.

We defined weak, strong, and dynamic controllability of CCNDs as two-player games. We classified the computational complexity of these games and discuss strategy synthesis algorithms. Weak controllability is Π_2^p-complete. Strong controllability is NP-complete. Dynamic controllability is PSPACE-complete.

We considered business processes with access control and conditional branches that may be both controllable and uncontrollable. We provided an encoding from this kind of processes into CCNDs. We mapped tasks and gateways to variables. We mapped mutual exclusive gateways and their associated Boolean variables to either observation or decision variables depending on if we desire to model controllable or uncontrollable outgoing branches. We mapped the partial order to order edges and authorization constraints to constraint edges. We mapped authorized users to the domains of the variables. We also showed that we can achieve further constraints on the allowed combinations of decisions.

We completely rewrote ZETA which is now a tool for CCNDs. We discussed a motivating example that we encoded into a CCND, validated and executed with ZETA with respect to the three kinds of controllability. We carried out an experimental evaluation against previous research and provided a new one for CCNDs. Overall, the new version of ZETA results faster. Finally, we also pointed out technical and non-technical limitations of our approach, which will provide the basis for the future work.

Acknowledgements This work was partially supported by MIUR, Project *Italian Outstanding Departments, 2018–2022*, and by INdAM, GNCS 2020, Projects *Strategic Reasoning and Automated Synthesis of Multi-Agent Systems* and *Automated Reasoning about Time in Medical and Business Applications*.

Funding Open access funding provided by Universitá degli Studi di Verona within the CRUI-CARE Agreement.

References

1. Amilhastre J, Fargier H, Marquis P (2002) Consistency restoration and explanations in dynamic csps application to configuration. Artif Intell 135(1):199–234
2. Bertino E, Bonatti PA, Ferrari E (2001) TRBAC: a temporal role-based access control model. ACM Trans Inf Syst Secur 4(3):191–233
3. Business process modeling notation 2.0. http://www.omg.org/spec/BPMN/2.0/
4. Brucker AD (2014) Using securebpmn for modelling security-aware service compositions. In: Secure and trustworthy service composition, lecture notes in computer science, vol 8900. Springer, pp 110–120
5. Brucker AD, Hang I, Lückemeyer G, Ruparel R (2012) SecureBPMN: modeling and enforcing access control requirements in business processes. In: SACMAT. ACM, pp 123–126
6. Cabanillas C, Resinas M, del-Río-Ortega A, Cortés AR (2015) Specification and automated design-time analysis of the business process human resource perspective. Inf Syst 52:55–82. https://doi.org/10.1016/j.is.2015.03.002
7. Cairo M, Combi C, Comin C, Hunsberger L, Posenato R, Rizzi R, Zavatteri M (2017) Incorporating decision nodes into conditional simple temporal networks. In: TIME 2017, vol. 90. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, pp 9:1–9:17. https://doi.org/10.4230/LIPIcs.TIME.2017.9
8. Combi C, Gambini M, Migliorini S, Posenato R (2014) 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):1182–1203. https://doi.org/10.1109/TSMC.2014.2300055
9. Combi C, Posenato R (2009) Controllability in temporal conceptual workflow schemata. In: BPM. Springer, pp 64–79
10. Combi C, Posenato R, Viganò L, Zavatteri M (2017) Access controlled temporal networks. In: ICAART 2017. ScitePress. https://doi.org/10.5220/0006185701180131
11. Combi C, Posenato R, Viganò L, Zavatteri M (2019) Conditional simple temporal networks with uncertainty and resources. J Artif Intell Res 64:931–985. https://doi.org/10.1613/jair.1.11453
12. Combi C, Viganò L, Zavatteri M (2016) Security constraints in temporal role-based access-controlled workflows. In: CODASPY 2016. ACM. https://doi.org/10.1145/2857705.2857716
13. Corradini F, Fornari F, Polini A, Re B, Tiezzi F (2019) Reprository: a repository platform for sharing business process models. In: Depaire B, Smedt JD, Dumas M, Fahland D, Kumar A, Leopold H, Reichert M, Rinderle-Ma S, Schulte S, Seidel S, van der Aalst WMP (eds) Proceedings of the dissertation award, doctoral consortium, and demonstration track at BPM 2019 co-located with 17th international conference on business process management, BPM 2019, Vienna, Austria, September 1-6, 2019, *CEUR Workshop Proceedings*, vol 2420, pp 149–153. CEUR-WS.org. http://ceur-ws.org/Vol-2420/paperDT7.pdf
14. Dechter R (2003) Constraint processing. Elsevier, Amsterdam
15. Dechter R, Dechter A (1988) Belief maintenance in dynamic constraint networks. In: 7th AAAI national conference on artificial intelligence, AAAI'88. AAAI Press, pp 37–42 (1988)
16. Dechter R, Meiri I, Pearl J (1991) Temporal constraint networks. Artif Intell 49(1–3):61–95
17. Dechter R, Pearl J (1987) Network-based heuristics for constraint-satisfaction problems. Artif Int. https://doi.org/10.1016/0004-3702(87)90002-6
18. Dumas M, Rosa ML, Mendling J, Reijers HA (2018) Fundamentals of business process management. Springer, Berlin. https://doi.org/10.1007/978-3-662-56509-4
19. Eder J, Franceschetti M, Köpke J (2018) Controllability of orchestrations with temporal SLA: encoding temporal XOR in CSTNUD. In: iiWAS 2018. ACM , pp 234–2422018). https://doi.org/10.1145/3282373.3282398
20. Fargier, H., Lang, J.: Uncertainty in constraint satisfaction problems: a probabilistic approach. In: Symbolic and quantitative approaches to reasoning and uncertainty, pp 97–104. Springer (1993)
21. Fargier H, Lang J, Martin-Clouaire R, Schiex T (1995) A constraint satisfaction framework for decision under uncertainty. In: 11th annual conference on uncertainty in artificial intelligence (UAI '95), pp 167–174. Morgan Kaufmann Publishers Inc
22. Fargier, H., Lang, J., Schiex, T.: Mixed Constraint Satisfaction: A Framework for Decision Problems Under Incomplete Knowledge. In: 13th National Conference on Artificial Intelligence - Volume 1, AAAI'96, pp. 175–180. AAAI Press (1996)
23. Franceschetti M, Eder J (2019) Towards checking dynamic controllability of processes with temporal loops. pp 1–14
24. Freuder EC (1978) Synthesizing constraint expressions. Commun ACM 21(11):958–966
25. Freuder EC (1982) A sufficient condition for backtrack-free search. J ACM 29:24–32
26. Gottlob G (2012) On minimal constraint networks. Artif Intell. https://doi.org/10.1016/j.artint.2012.07.006
27. Hunsberger L, Posenato R, Combi C (2012) The dynamic controllability of conditional STNs with uncertainty. In: PlanEx at ICAPS-2012. arXiv:1212.2005
28. Hunsberger L, Posenato R, Combi C (2015) A sound-and-complete propagation-based algorithm for checking the dynamic consistency of conditional simple temporal networks. In: 22nd international symposium on temporal representation and reasoning (TIME 2015), pp 4–18. IEEE Computer Society
29. Mackworth AK (1977) Consistency in networks of relations. Artif Intell 8(1):99–118
30. Milosavljevic G, Sladic G, Milosavljevic B, Zaric M, Gostojic S, Slivka J (2018) Context-sensitive constraints for access control of business processes. Comput Sci Inf Syst 15(1):1–30
31. Mittal S, Falkenhainer B (1990) Dynamic constraint satisfaction problems. In: 8th national conference on artificial intelligence, AAAI'90, pp 25–32. AAAI Press
32. Mohr R, Henderson TC (1986) Arc and path consistency revisited. Artif Intell 28(2):225–233
33. Montanari U (1974) Networks of constraints: fundamental properties and applications to picture processing. Sci Inf 7:95–132
34. Morris PH, Muscettola N, Vidal T (2001) Dynamic control of plans with temporal uncertainty. IJCAI 2001:494–502
35. de Moura L, Bjørner N (2008) Z3: an efficient smt solver. In: Tools and algorithms for the construction and analysis of systems, pp 337–340. Springer

36. Nouioua M, Zouari B, Alti A (2019) Formal approach for authorization in distributed business process related task document role based access control. In: IWCMC, pp 1964–1970. IEEE
37. Posenato R, Zerbato F, Combi C (2018) Managing decision tasks and events in time-aware business process models. In: BPM 2018, *LNCS*, vol 11080, pp 102–118. Springer. https://doi.org/10.1007/978-3-319-98648-7_7
38. Ramadan Q, Strüber D, Salnitri M, Jürjens J, Riediger V, Staab S (2020) A semi-automated bpmn-based framework for detecting conflicts between security, data-minimization, and fairness requirements. Softw Syst Model 19(5):1191–1227
39. Rosa ML, Dumas M, ter Hofstede AHM, Mendling J, Gottschalk F (2008) Beyond control-flow: extending business process configuration to roles and objects. In: ER, *Lecture Notes in Computer Science*, vol 5231, pp 199–215. Springer
40. Sabin M, Freuder EC (1999) Detecting and resolving inconsistency and redundancy in conditional constraint satisfaction problems. AAAI Technical Report
41. Sandhu RS, Coyne EJ, Feinstein HL, Youman CE (1996) Role-based access control models. Computer 29(2):38–47. https://doi.org/10.1109/2.485845
42. Schefer-Wenzl S, Strembeck M (2013) Modelling context-aware RBAC models for mobile business processes. Int J Wirel Mob Comput 6(5):448–462
43. Tsamardinos I, Vidal T, Pollack ME (2003) CTP: a new constraint-based formalism for conditional. Temp Plann Constr 8(4):365–388
44. Vidal T, Fargier H (1999) Handling contingency in temporal constraint networks: from consistency to controllabilities. J Exp Theor Artif Intell 11(1):23–45
45. Wang Q, Li N (2010) Satisfiability and resiliency in workflow authorization systems. ACM Trans Inf Syst Secur 13(4)40:1–40:35
46. Weske M (2012) Business process management-concepts, languages, architectures, 2nd edn. Springer, Berlin. https://doi.org/10.1007/978-3-642-28616-2
47. Weske M, van der Aalst WMP, Verbeek HMW (2004) Advances in business process management. Data Knowl Eng 50(1):1–8. https://doi.org/10.1016/j.datak.2004.01.001
48. Weske M, Decker G, Dumas M, La Rosa M, Mendling J, Reijers HA (2020) Model collection of the business process management academic initiative. https://doi.org/10.5281/zenodo.3758705
49. Zavatteri M (2017) Conditional simple temporal networks with uncertainty and decisions. In: TIME 2017, vol 90, pp 23:1–23:17. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. https://doi.org/10.4230/LIPIcs.TIME.2017.23
50. Zavatteri M (2019) Temporal and resource controllability of workflows under uncertainty. In: Proceedings of the dissertation award, demonstration, and industrial track at BPM 2019, vol 2420, pp 9–14. CEUR-WS.org. http://ceur-ws.org/Vol-2420/paperDA3.pdf
51. Zavatteri M, Combi C, Posenato R, Viganò L (2017) Weak, strong and dynamic controllability of access-controlled workflows under conditional uncertainty. In: BPM 2017, pp 235–251. Springer. https://doi.org/10.1007/978-3-319-65000-5_14
52. Zavatteri M, Combi C, Rizzi R, Viganò L (2019) Hybrid SAT-based consistency checking algorithms for simple temporal networks with decisions. In: TIME 2019, vol 147, p 2:12:17. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. https://doi.org/10.4230/LIPIcs.TIME.2019.2
53. Zavatteri M, Combi C, Rizzi R, Viganò L (2020) Consistency checking of STNs with decisions: managing temporal and access-control constraints in a seamless way. Information and Computation (in-press). https://doi.org/10.1016/j.ic.2020.104637. http://www.sciencedirect.com/science/article/pii/S0890540120301255
54. Zavatteri M, Combi C, Viganò L (2019) Resource controllability of workflows under conditional uncertainty. In: Business process management workshops, pp 68–80. Springer. https://doi.org/10.1007/978-3-030-37453-2_7
55. Zavatteri M, Rizzi R, Villa T (2019) Complexity of weak, strong and dynamic controllability of CNCUs. In: First workshop on formal verification, logic, automata, and synthesis, 2019, OVERLAY 2019, vol 2509, pp 83–88. CEUR-WS.org. http://ceur-ws.org/Vol-2509/paper13.pdf
56. Zavatteri M, Rizzi R, Villa T (2019) Strong controllability of temporal networks with decisions. In: First workshop on formal verification, logic, automata, and synthesis, 2019, OVERLAY 2019, vol 2509, pp 77–82. CEUR-WS.org. http://ceur-ws.org/Vol-2509/paper12.pdf
57. Zavatteri M, Rizzi R, Villa T (2020) Dynamic controllability and (J, K)-resiliency in generalized constraint networks with uncertainty. In: Proceedings of the thirtieth international conference on automated planning and scheduling, ICAPS 2020, pp 314–322. AAAI Press
58. Zavatteri M, Rizzi R, Villa T (2020) On the complexity of resource controllability in business process management. In: Business process management workshops (to appear). Springer
59. Zavatteri M, Viganò L (2018) Constraint networks under conditional uncertainty. In: ICAART 2018, pp 41–52. SciTePress. https://doi.org/10.5220/0006553400410052
60. Zavatteri M, Viganò L (2019) Conditional simple temporal networks with uncertainty and decisions. Theor Comput Sci 797:77–101. https://doi.org/10.1016/j.tcs.2018.09.023
61. Zavatteri M, Viganò L (2019) Conditional uncertainty in constraint networks. In: Agents and artificial intelligence, pp 130–160. Springer. https://doi.org/10.1007/978-3-030-05453-3_7
62. Zavatteri M, Viganò L (2019) Last man standing: static, decremental and dynamic resiliency via controller synthesis. J Comput Secur 27(3):343–373. https://doi.org/10.3233/JCS-181244