Constraint Networks Under Conditional Uncertainty

Matteo Zavatteri and Luca Viganò
Department of Computer Science, University of Verona, Verona, Italy
Department of Informatics, King's College London, London, U.K.

Keywords: Constraint Networks, Conditional Uncertainty, Controllability, Resource Scheduling, AI-based Security, CNCU.

Abstract: *Constraint Networks (CNs)* are a framework to model the *constraint satisfaction problem (CSP)*, which is the problem of finding an assignment of values to a set of variables satisfying a set of given constraints. Therefore, CSP is a satisfiability problem. When the CSP turns conditional, consistency analysis extends to finding also an assignment to these conditions such that the relevant part of the initial CN is consistent. However, CNs fail to model CSPs expressing an *uncontrollable* conditional part (i.e., a conditional part that cannot be decided but merely observed as it occurs). To bridge this gap, in this paper we propose *constraint networks under conditional uncertainty (CNCUs)*, and we define weak, strong and dynamic *controllability* of a CNCU. We provide algorithms to check each of these types of controllability and discuss how to synthesize (dynamic) execution strategies that drive the execution of a CNCU saying which value to assign to which variable depending on how the uncontrollable part behaves. We benchmark the approach by using ZETA, a tool that we developed for CNCUs. What we propose is fully automated from analysis to simulation.

1 INTRODUCTION

Context and Motivations. Assume that we are given a resource-scheduling problem specifying a conditional part that is out of control, and that we are then asked to schedule (some of the) resources in a way that meets all relevant constraints, or to prove that such a scheduling does not exist. We are also permitted to make our scheduling decisions as we like. In general, we can act in three different main ways:

1. We assume that we can predict the future and then make sure that a (possibly different) scheduling for each possible uncontrollable behavior exists.
2. We assume that we know nothing and then make sure that at least a single solution always works.
3. We assume that we can make our scheduling decisions according to what is going on around us.

These are the intuitions behind the three main kinds of *controllability*: *weak* (for presumptuous), *strong* (for anxious) and *dynamic* (for grandmasters).

In recent years, a considerable amount of research has been carried out to investigate controllability analysis in order to deal with temporal and conditional uncertainty, either in isolation or simultaneously. In particular, a number of extensions of *simple temporal networks* (*STNs*, (Dechter et al., 1991)) have been proposed. For example, *simple temporal networks with uncertainty* (*STNUs*, (Morris et al., 2001)) add uncontrollable (but bounded) durations between pairs of temporal events, whereas *conditional simple temporal networks* (*CSTNs*, (Hunsberger et al., 2015)) and formerly conditional temporal problem (*CTP*, (Tsamardinos et al., 2003)) extend STNs by turning the constraints conditional. Finally, *conditional simple temporal networks with uncertainty* (*CSTNUs*, (Hunsberger et al., 2012)) merge STNUs and CSTNs, whereas *conditional simple temporal networks with uncertainty and decisions* (*CSTNUDs*, (Zavatteri, 2017)) encompass all previous formalisms.

Several algorithms have been proposed to check the controllability of a temporal network, e.g., constraint-propagation (Hunsberger et al., 2015), timed game automata (Cimatti et al., 2016; Zavatteri, 2017) and satisfiability modulo theory (Cimatti et al., 2015a; Cimatti et al., 2015b).

Research has also been carried out in the "discrete" world of classic *constraint networks* (*CNs*) (Dechter, 2003) in order to address different kinds of uncertainty. For example, a *mixed constraint satisfaction problem* (*Mixed CSP*, (Fargier et al., 1996)) divides the set of variables in controllable and uncon-

Zavatteri, M. and Viganò, L.
Constraint Networks Under Conditional Uncertainty.
DOI: 10.5220/0006553400410052
In *Proceedings of the 10th International Conference on Agents and Artificial Intelligence (ICAART 2018) - Volume 2*, pages 41-52
ISBN: 978-989-758-275-2
Copyright © 2018 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

trollable, whereas a *dynamic constraint satisfaction problem* (*DCSP*, (Mittal and Falkenhainer, 1990)) introduces activity constraints saying when variables are relevant depending on what values some other variables have been assigned. Probabilistic approaches such as (Fargier and Lang, 1993) aim instead at finding the most probable working solution.

Despite all this, a formal model to extend classic CNs (Dechter, 2003) with conditional uncertainty adhering to the modeling ideas employed by CSTNs is still missing. In a CSTN, for instance, time points (variables) and linear inequalities (constraints) are labeled by conjunctions of literals where the truth value assignments to the embedded Boolean propositions are out of control. Every proposition has an associated *observation time point*, a special kind of time point that reveals the truth value assignment to the associated proposition upon its execution (i.e., as soon as it is assigned a real value). Equivalently, this truth value assignment can be thought of as being under the control of the environment.

Contributions. Our contributions in this paper are three-fold. First, we define *constraint networks under conditional uncertainty (CNCUs)* as an extension of classic CNs and we give the semantics for weak, strong and dynamic controllability. Second, we provide algorithms for each of these types of controllability. Third, we discuss ZETA, a prototype tool that we developed to automate and benchmark our results.

Organization. Section 2 provides essential background on CNs and the adaptive consistency algorithm. Section 3 introduces a motivational example. Section 4 introduces our main contribution: CNCUs. Section 5 defines the semantics for weak, strong and dynamic controllability and Section 6 addresses the related algorithms. Section 7 discusses our tool ZETA for CNCUs along with an experimental evaluation. Section 8 discusses the correctness of our approach. Section 9 discusses related work. Section 10 draws conclusions and discusses future work.

2 BACKGROUND

In this section, we briefly review CNs and the *adaptive consistency algorithm* for the related consistency checking (Dechter, 2003). We renamed some symbols for coherence with the rest of the paper.

Definition 1. A *Constraint Network (CN)* is a triple $\mathcal{Z} = \langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{V} = \{V_1, \ldots, V_n\}$ is a finite set of variables, $\mathcal{D} = \{D_1, \ldots, D_n\}$ is a set of discrete domains $D_i = \{v_1, \ldots, v_j\}$ (one for each variable), and $\mathcal{C} = \{R_{S_1}, \ldots, R_{S_n}\}$ is a finite set of constraints each one represented as a relation R_S defined over a *scope* of variables $S \subseteq \mathcal{V}$ such that if $S = \{V_i, \ldots, V_r\}$, then $R \subseteq D_i \times \cdots \times D_r$. A CN is *consistent* if each variable $V_i \in \mathcal{V}$ can be assigned a value $v_i \in D_i$ such that *all* constraints are satisfied. □

Algorithm 1: ADC$(\mathcal{Z}, d)$.

```
Input: A CN $\mathcal{Z} = \langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$ and an ordering $d = V_1 \prec \cdots \prec V_n$
Output: A set Buckets of buckets (one for each variable) if $\mathcal{Z}$ is
        consistent, inconsistent otherwise.
for $i \leftarrow n$ downto 1 do        ▷ Partition the constraints as follows:
    Put in $Bucket(V_i)$ all unplaced constraints mentioning $V_i$
for $p \leftarrow n$ downto 1 do
    Let $j \leftarrow |Bucket(V_p)|$ and $S_i$ be the scope of
      $R_{S_i} \in Bucket(V_p)$
    $S' \leftarrow \bigcup_{i=1}^{j} S_i \setminus \{V_p\}$
    $R_{S'} \leftarrow \pi_{S'}(\bowtie_{i=1}^{j} R_{S_i})$
    if $R_{S'} \neq \emptyset$ then
        $Bucket(V') \leftarrow Bucket(V') \cup \{R_{S'}\}$, where $V' \in S'$ is
          the "latest" variable in $d$.
    else
        return inconsistent
$Buckets = \{\{Bucket(V)\} \mid V \in \mathcal{V}\}$
return Buckets
```

The *constraint satisfaction problem* (*CSP*) is NP-hard (Dechter, 2003). A CN is k-ary if all constraints have scope cardinality $\leq k$, *binary* when $k = 2$ (Dechter, 2003; Montanari, 1974).

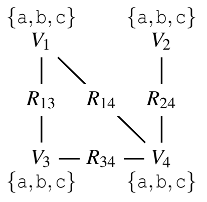
Let R_{ij} be a shortcut to represent a binary relation having scope $S = \{V_i, V_j\}$. A binary CN is *minimal* if any tuple $(v_i, v_j) \in R_{ij} \in \mathcal{C}$ belongs to at least one global solution for the underlying CSP (Montanari, 1974). Besides for a few restricted classes of CNs, the general process of computing a minimal network is NP-hard (Montanari, 1974). Furthermore, even considering a binary minimal network, the problem of generating an arbitrary solution is NP-hard if there is no total order on the variables (Gottlob, 2012).

Therefore, a first crude technique is that of searching for a solution by exhaustively enumerating (and testing) all possible solutions and stopping as soon as one satisfies all constraints in $\mathcal{C}$. To speed up the search, we can combine techniques such as backtracking with pruning techniques such as *node*, *arc* and *path consistency* (Mackworth, 1977).

k-consistency guarantees that any (locally consistent) assignment to any subset of $(k-1)$-variables can be extended to a k^{th} (still unassigned) variable such that all constraints between these k-variables are satisfied. *Strong k-consistency* is k-consistency for each j such that $1 \leq j \leq k$ (Freuder, 1982).

Directional consistency has been introduced to speed up the process of synthesizing a solution for a constraint network limiting backtracking (Dechter and Pearl, 1987). In a nutshell, given a total order

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS



(a) Constraint graph.

$Bucket(V_4) : R_{14}, R_{24}, R_{34} \|$

$Bucket(V_3) : R_{13} \| R_{123}$

$Bucket(V_2) : \| R_{12}$

$Bucket(V_1) : \| R_{1}$

(b) $R_{ij} \in \mathcal{C}$

Figure 1: Graphical representation of a binary CN.

on the variables of a CN, the network is *directional-consistent* if it is consistent with respect to the given order that dictates the assignment order of variables. In (Dechter and Pearl, 1987), an *adaptive-consistency (*ADC*) algorithm* was provided as a directional consistency algorithm adapting the level of k-consistency needed to guarantee a backtrack-free search once the algorithm terminates, if the network is consistent (see Algorithm 1). The input of ADC is a CN $\mathcal{Z} = \langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$ along with an order d for $\mathcal{V}$. At each step the algorithm *adapts* the level of consistency to guarantee that if the network passes the test, any solution satisfying all constraints can be found without backtracking. If the network is inconsistent, the algorithm detects it before the solution generation process starts. ADC initializes a $Bucket(V)$ for each variable $V \in \mathcal{V}$ and first processes all the variables top-down (i.e., from last to first following the ordering d) by filling each bucket with all (still unplaced) constraints $R_S \in \mathcal{C}$ such that $V \in S$. Then, it processes again the variables top-down and, for each variable V, it computes a new scope S' consisting of the union of all scopes of the relations in $Bucket(V)$ neglecting V itself. After that, it computes a new relation $R_{S'}$ by joining all $R_S \in Bucket(V)$ and projecting with respect to S' ($\bowtie$ and π are the join and projection operators of relational algebra). In this way, it enforces the appropriate level of consistency. If the resulting relation is empty, then $\mathcal{Z}$ is inconsistent; otherwise, the algorithm adds $R_{S'}$ to the bucket of the *latest* variable in S' (with respect to the ordering d), and goes on with the next variable. Finally, it returns the set of *Buckets* (we slightly modified the return statement of ADC).

Any binary CN can be represented as a *constraint graph* where the set of nodes coincides with $\mathcal{V}$ and the set of edges represents the constraints in $\mathcal{C}$. Furthermore, nodes are labeled by their domains. Each (undirected) edge between two variables V_1 and V_2 is labeled by the corresponding $R_{12} \in \mathcal{C}$. As an example, consider the *constraint graph* in Figure 1(a) representing $\mathcal{Z} = \langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{V} = \{V_1, V_2, V_3, V_4\}$, $\mathcal{D} = \{D_1, D_2, D_3, D_4\}$, with $D_1 = D_2 = D_3 = D_4 = \{\texttt{a}, \texttt{b}, \texttt{c}\}$, and $\mathcal{C} = \{R_{13}, R_{14}, R_{24}, R_{34}\}$. All $R_{ij} \in \mathcal{C}$ contain the same tuples; actually, they all specify the $\neq$ constraint between the pair of variables they connect. That is, $R_{13} = R_{14} = R_{24} = R_{34} = \{(\texttt{a},\texttt{b}), (\texttt{a},\texttt{c}), (\texttt{b},\texttt{a}), (\texttt{b},\texttt{c}), (\texttt{c},\texttt{a}), (\texttt{c},\texttt{b})\}$.

The CN in Figure 1(a) is consistent. To prove that we chose, without loss of generality (recall that *any* order is fine for this algorithm (Dechter, 2003)), the order $d = V_1 \prec V_2 \prec V_3 \prec V_4$ and ran $\text{ADC}(\mathcal{Z}, d)$. The output of the algorithm is shown in Figure 1(b). ADC first processes V_4 by filling $Bucket(V_4)$ with R_{14}, R_{24} and R_{34} (as they all mention V_4 in their scope and are still unplaced). Then, it processes V_3 by filling $Bucket(V_3)$ with R_{13} (but not R_{34}). Finally, it leaves $Bucket(V_2)$ and $Bucket(V_1)$ empty as all relations mentioning V_2 and V_1 in their scope have already been put in some other buckets. Therefore, the initialization phase fills the buckets in Figure 1(b) with all relations on the left of $\|$ (the newly generated ones will appear on the right).

In the second phase, the algorithm computes $R_{123} = \pi_{123}(R_{14} \bowtie R_{24} \bowtie R_{34}) = \{(\texttt{a},\texttt{a},\texttt{a}), (\texttt{a},\texttt{a},\texttt{b}), (\texttt{a},\texttt{a},\texttt{c}), (\texttt{a},\texttt{b},\texttt{a}), (\texttt{a},\texttt{b},\texttt{b}), (\texttt{a},\texttt{c},\texttt{a}), (\texttt{a},\texttt{c},\texttt{c}), (\texttt{b},\texttt{a},\texttt{a}), (\texttt{b},\texttt{a},\texttt{b}), (\texttt{b},\texttt{b},\texttt{a}), (\texttt{b},\texttt{b},\texttt{b}), (\texttt{b},\texttt{b},\texttt{c}), (\texttt{b},\texttt{c},\texttt{b}), (\texttt{b},\texttt{c},\texttt{c}), (\texttt{c},\texttt{a},\texttt{a}), (\texttt{c},\texttt{a},\texttt{c}), (\texttt{c},\texttt{b},\texttt{b}), (\texttt{c},\texttt{b},\texttt{c}), (\texttt{c},\texttt{c},\texttt{a}), (\texttt{c},\texttt{c},\texttt{b}), (\texttt{c},\texttt{c},\texttt{c})\}$ and adds it to $Bucket(V_3)$ (the latest variable in the scope $\{V_1, V_2, V_3\}$). Then, it goes ahead by processing $Bucket(V_3)$ generating in a similar way R_{12} and adding it to $Bucket(V_2)$. Finally, it processes $Bucket(V_2)$ by computing R_1 and adding it to $Bucket(V_1)$. Since the joins yielded no empty relation, it follows that $\mathcal{Z}$ is consistent.

We generate a solution by assigning the variables following the order d. For each $V \in d$ we just look for a value v in its domain such that the current solution augmented with $V = v$ satisfies all constraints in $Bucket(V)$. If the network is consistent, at least one value is guaranteed to be there. In this way, each solution can be generated efficiently without backtracking by assigning one variable at a time. A possible solution is $V_1 = \texttt{a}$, $V_2 = \texttt{c}$, $V_3 = \texttt{c}$ and $V_4 = \texttt{b}$.

3 MOTIVATIONAL EXAMPLE

As a motivational example, we consider an adaptation of a standard workflow/business-process example that describes a *loan origination process* (*LOP*) for eligible customers whose financial records have already been approved. We tuned the example in order to focus on a few characteristics of interest.

The Workflow. The workflow follows a free composition approach and is subject to conditional uncertainty as the truth value assignments to the Boolean `pers?` and `rnd?` are *out of control* (Figure 2).

The LOP starts by processing a request (`ProcReq`)

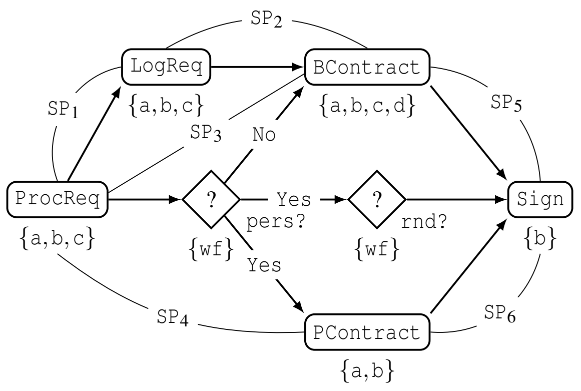


Figure 2: A simplification of a loan origination process. ProcReq, LogReq, Sign and the leftmost conditional split connector are always executed. PContract and the rightmost conditional split connector are executed iff pers? $=$ T. BContract is executed iff pers? $=$ F. rnd? only influences security policies. We abbreviate Alice, Bob, Charlie, David and workflow engine to a, b, c, d and wf.

with Alice, Bob and Charlie being the only authorized users. After that, the request is logged for future accountability purposes (LogReq) with the same users of ProcReq authorized for this task. The flow of execution then splits into two (mutually-exclusive) branches upon the execution of the first conditional split connector (leftmost diamond labeled by ?) which sets the truth value of pers? according to the *discovered* type of loan. A workflow engine wf is authorized to execute this split connector.

If pers? is true (T), it means that the workflow will handle a *personal loan* and that the flow of the execution continues by preparing a personal contract (PContract), with Alice and Bob the only authorized users. Moreover, when processing personal loans, different security policies hold depending on what truth value a second Boolean variable (rnd?) is assigned (see below). The truth value of rnd? is generated at random upon the execution of the second conditional split connector (rightmost diamond) whose authorized user is again wf. Thus, the truth value assignment of rnd? can be thought of as uncontrollable as well. Note that no task will be prevented from executing depending on the value of rnd?, only the users carrying them out will (see the end of this section).

Instead, if pers? is false (F), the workflow will handle a *business loan* and the flow of execution continues by preparing a business contract (BContract) with Alice, Bob, Charlie and David authorized users.

Finally, regardless of the truth values of pers? and rnd? the LOP concludes with the signing of the contract (Sign) with Bob the only authorized user.

Security Policies. A *separation of duties* (SoD) (resp., *binding of duties (BoD)*) between two tasks T_1 and T_2 says that the users executing T_1 and T_2 must be different (resp., equal). In our example, Alice and Bob are married and thus the only relatives. Our process enforces six security policies (SP$_1$ $-$ SP$_6$).

SP$_1$ calls for a SoD between ProcReq and LogReq *and* also requires that the users executing the two tasks must not be relatives if pers? $=$ F. SP$_2$ calls for a SoD between LogReq and BContract (implicitly when pers? $=$ F). SP$_3$ calls for a SoD between ProcReq and BContract (implicitly when pers? $=$ F). SP$_4$ calls for a SoD between ProcReq and PContract *and* also requires that the users executing the two tasks must not be relatives (implicitly when pers? $=$ T). SP$_5$ calls for a SoD between BContract and Sign (implicitly when pers? $=$ F). SP$_6$ calls for *either* a SoD between PContract and Sign if pers? $=$ T and rnd? $=$ T, *or* a BoD between PContract and Sign if pers? $=$ T and rnd? $=$ F.

4 CNCUs

In this section, we extend CNs to address conditional uncertainty. We call this new kind of network *Constraint Network under Conditional Uncertainty (CNCU)*. CNCUs are obtained by extending CNs 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 the environment),
- *observation variables* to observe such truth value assignments, and
- *labels* to enable or disable a subset of variables and constraints, and therefore introduce a (implicit) notion of partial order among the variables.

We will also talk about *execution* meaning that we *execute a variable* by assigning it a value and we *execute a CNCU* by executing all relevant variables. Variables and constraints are *relevant* if they must be considered during execution.

Given a set $\mathcal{P} = \{p, q, \dots\}$ of Boolean propositions, a *label* $\ell = l_1 \land \cdots \land l_n$ is a finite conjunction of literals l_i, where a literal is either a proposition $p \in \mathcal{P}$ (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 labels drawn from $\mathcal{P}$; e.g., if $\mathcal{P} = \{p, q\}$, then $\mathcal{P}^* = \{\boxdot, p, q, \neg p, \neg q, p \land q, p \land \neg q, \neg p \land q, \neg p \land \neg q, p \land \neg p, q \land \neg q\}$. A label $\ell_1 \in \mathcal{P}^*$ is *consistent* iff ℓ_1 is satisfiable, *entails* a label ℓ_2 (written $\ell_1 \Rightarrow \ell_2$) iff all literals in ℓ_2 appear in ℓ_1 too (i.e., if ℓ_1 is more *specific* than ℓ_2) and *falsifies* a label ℓ_2 iff $\ell_1 \land \ell_2$ is not consistent. The *difference* of two labels ℓ_1 and

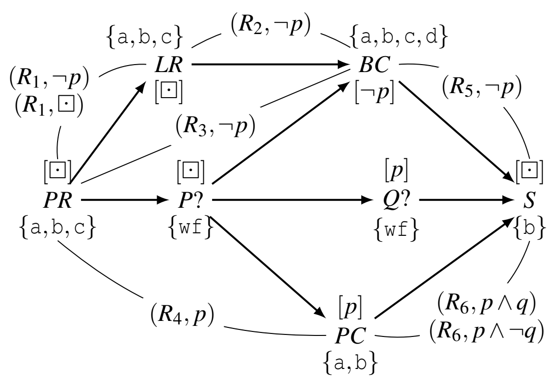


Figure 3: Binary CNCU modeling the workflow in Figure 2.

ℓ_2 is a new label $\ell_3 = \ell_1 - \ell_2$ consisting of all literals of ℓ_1 minus those shared with ℓ_2. For instance, if $\ell_1 = p \wedge \neg q$ and $\ell_2 = p$, then ℓ_1 and ℓ_2 are consistent, $\ell_1 \Rightarrow \ell_2$, $\ell_1 - \ell_2 = \neg q$ and $\ell_2 - \ell_1 = \boxdot$.

Definition 2. A *constraint network under conditional uncertainty (CNCU)* is a tuple $\langle \mathcal{V}, \mathcal{D}, D, \mathcal{OV}, \mathcal{P}, O, L, \prec, \mathcal{C}\rangle$, where:

- $\mathcal{V} = \{V_1, V_2, \dots\}$ is a finite set of variables.
- $\mathcal{D} = \{D_1, D_2, \dots\}$ is a set of discrete domains.
- $D\colon \mathcal{V} \to \mathcal{D}$ is a mapping assigning a domain to each variable.
- $\mathcal{OV} \subseteq \mathcal{V} = \{P?, Q?, \dots\}$ is a set of *observation variables*.
- $\mathcal{P} = \{p, q, \dots\}$ is a set of Boolean *propositions* whose truth values are all initially unknown.
- $O\colon \mathcal{P} \to \mathcal{OV}$ is a bijection assigning a unique observation variable $P?$ to each proposition p. When $P?$ executes, the truth value of p becomes known and no longer changes.
- $L\colon \mathcal{V} \to \mathcal{P}^*$ is a mapping assigning a label ℓ to each variable V saying when V is relevant.
- $\prec \subseteq \mathcal{V} \times \mathcal{V}$ is a precedence relation on the variables. We write $(V_1, V_2) \in \prec$ (or $V_1 \prec V_2$) to express that V_1 is assigned *before* V_2.
- $\mathcal{C}$ is a finite set of *labeled constraints* of the form (R_S, ℓ), where $S \subseteq \mathcal{V}$ and $\ell \in \mathcal{P}^*$. If $S = \{V_1, \dots, V_n\}$, then $R_S \subseteq D(V_1) \times \cdots \times D(V_n)$. $\square$

We graphically represent a (binary) CNCU by extending the constraint graph discussed for CNs into a *labeled constraint (multi)graph*, where each variable is also labeled by its label $L(V)$, and the edges are of two kinds: *order edges* (directed unlabeled edges) and *constraint edges* (undirected labeled edges). An order edge $V_1 \to V_2$ models $V_1 \prec V_2$. A constraint edge between V_1 and V_2 models (R_{12}, ℓ). Many constraint edges may possibly be specified between the same pair of variables, as long as ℓ is different (e.g., $(R_1, \boxdot)$ and $(R_1, \neg p)$ between PR and LR in Figure 3).

Figure 3 shows a CNCU modeling the workflow in Figure 2: PR, LR, PC BC and S model `ProcReq`, `LogReq`, `PContract`, `BContract` and `Sign`, whereas $P?$ and $Q?$ (observation variables) model the two conditional split connectors. $P?$ and $Q?$ are associated to the Boolean propositions p and q which, in turn, abstract `pers?` and `rnd?`. The union of all relations having the form (R_i, ℓ) model the security policy $\texttt{SP}_i$ discussed at the end of Section 3. We show all constraints of Figure 3 in Figure 4. For example, $(R_1, \boxdot)$ (Figure 4(a)) contains all tuples (x, y) such that $x \neq y$ (what must *always* hold) and $(R_1, \neg p)$ (Figure 4(b)) contains all tuples (x, y) such that x and y are not relatives (what must *also* hold for business loans). Thus, $(R_1, \boxdot)$ and $(R_1, \neg p)$ model $\texttt{SP}_1$.

(a) $(R_1,\boxdot)$

PR	LR
a	b
a	c
b	a
b	c
c	a
c	b

(b) $(R_1,\neg p)$

PR	LR
a	a
a	c
b	b
b	c
c	a
c	b
c	c

(c) $(R_2,\neg p)$

LR	BC
a	b
a	c
a	d
b	a
b	c
b	d
c	a
c	b
c	d

(d) $(R_3,\neg p)$

PR	BC
a	b
a	c
a	d
b	a
b	c
b	d
c	a
c	b
c	d

(e) (R_4,p)

PR	PC
c	a
c	b

(f) $(R_5,\neg p)$

BC	S
a	b
c	b
d	b

(g) $(R_6,p\wedge q)$

PC	S
a	b

(h) $(R_6,p\wedge\neg q)$

PC	S
b	b

Figure 4: Labeled constraints of the CNCU in Figure 3.

In the rest of this section, we say when CNCUs are well-defined. We import the notions of label honesty and coherence from temporal networks (see, e.g., (Hunsberger et al., 2015; Zavatteri, 2017)).

A label ℓ labeling a variable or a constraint is *honest* if for each literal p or $\neg p$ in ℓ we have that $\ell \Rightarrow L(P?)$, where $P? = O(p)$ is the observation variable associated to p; ℓ is dishonest otherwise. For example, consider $(R_6, p \wedge q)$ in Figure 3. The constraint applies only if $p = q = \texttt{T}$. However, the truth value of q is set (by the environment) upon the execution of $Q?$, which in turn is relevant iff p was *previously* assigned true (as $L(Q?) = p$). Thus, an honest ℓ containing q or $\neg q$ should also contain p. A label on a constraint is *coherent* if it entails the labels of all variables in the scope of the constraint.

Definition 3. A CNCU $\langle \mathcal{V}, \mathcal{D}, D, \mathcal{OV}, \mathcal{P}, O, L, \prec, \mathcal{C}\rangle$ is *well defined* iff all labels are consistent and the following properties hold.

- *Variable Label Honesty.* $L(V)$ is honest for any $V \in \mathcal{V}$, and $O(p) \prec V$ for any p or $\neg p$ belonging to $L(V)$. That is, V only executes when the honest $L(V)$ becomes completely known and evaluates to true; e.g., *BC after P? if* $\neg p$ in Figure 3.

- *Constraint Label Honesty.* ℓ is honest for any $(R_S, \ell) \in \mathcal{C}$. That is, R_S only applies when the honest ℓ becomes completely known and evaluates to true; e.g., $(R_6, p \wedge q)$ in Figure 3 if *after* $P?$ and $Q?$, p and q *are observed* true.
- *Constraint Label Coherence.* $\ell \Rightarrow L(V)$ for any $(R_S, \ell) \in \mathcal{C}$ and any $V \in S$. That is, the label of a constraint is at least as specific as any label of the variables in its scope; e.g., $(R_6, p \wedge q)$ in Figure 3.
- *Precedence Relation Coherence.* For any $V_1, V_2 \in \mathcal{V}$, if $V_1 \prec V_2$ then $L(V_1) \wedge L(V_2)$ is consistent. That is, no partial order can be specified between variables not taking part together in any execution; e.g. PC and BC in Figure 3. $\square$

Thus, the CNCU in Figure 3 is well-defined.

5 SEMANTICS

We give the semantics for weak, strong and dynamic controllability of CNCUs. We note that in this paper we provide algorithms *relying on total orderings* so as to handle as many solutions as possible.

Definition 4. A *scenario* is a mapping $s\colon \mathcal{P} \to \{\mathtt{T}, \mathtt{F}, \mathtt{U}\}$ assigning *true* or *false* or *unknown* to each proposition in $\mathcal{P}$.

A scenario s is *honest* if for any $p \in \mathcal{P}$ where $s(p) \neq \mathtt{U}$, we have that $s(q) = \mathtt{T}$ for any $q \in L(O(p))$, and $s(q) = \mathtt{F}$ for any $\neg q \in L(O(p))$.

A scenario s *satisfies* a label ℓ (written $s \models \ell$) if s satisfies all literals in ℓ, where $s \models p$ iff $s(p) = \mathtt{T}$ (positive literal) and $s \models \neg p$ iff $s(p) = \mathtt{F}$ (negative literal).

An (honest) scenario s is *partial* if there exists an unknown proposition p (i.e., $s(p) = \mathtt{U}$) such that $s \models L(P?)$ (i.e., $P? = O(p)$ is relevant in s); s is *complete* otherwise. The *initial* scenario is that in which all propositions are unknown.

We write $\mathcal{S}$ to denote the set of all scenarios. $\square$

Consider Figure 3. If $s(p) = \mathtt{T}$ and $s(q) = \mathtt{U}$, then s is honest (note that $L(P?) = \boxdot$ so no check is required) and partial as $s(q) = \mathtt{U}$ and $Q?$ is relevant for s because $s \models L(Q?)$. Instead, if $s(p) = \mathtt{U}$, $s(q) = \mathtt{T}$, then s is dishonest as q can only be assigned upon the execution of $Q?$, which requires $s(p) = \mathtt{T}$. Moreover, $s_1(p) = \mathtt{T}$, $s_1(q) = \mathtt{T}$ and $s_2(p) = \mathtt{T}$, $s_2(q) = \mathtt{F}$ are both honest and complete (Figure 3).

From now on, we will assume scenarios to be honest, sometimes partial, sometimes complete, unless stated otherwise. In what follows, we give the definition of *projection*, an operation to turn a CNCU into a classic CN according to a *complete* scenario s, which is crucial to define the three kinds of controllability.

Definition 5. Let $\mathcal{Z} = \langle \mathcal{V}, \mathcal{D}, D, \mathcal{OV}, \mathcal{P}, O, L, \prec, \mathcal{C} \rangle$ be a CNCU and s any complete scenario. The *projection* of $\mathcal{Z}$ onto s is a CN $\mathcal{Z}_s = \langle \mathcal{V}_s, \mathcal{D}, \mathcal{C}_s \rangle$ such that:

- $\mathcal{V}_s = \{V \mid V \in \mathcal{V} \wedge s \models L(V)\}$
- $\mathcal{C}_s = \{R_S \mid (R_S, \ell) \in \mathcal{C} \wedge s \models \ell\}$ $\square$

For example the projection of Figure 3 with respect to the initial scenario $s(p) = \mathtt{U}$ and $s(q) = \mathtt{U}$ results in a CN, where $\mathcal{V}_s = \{PR, LR, P?, S\}$, and $\mathcal{C}_s = \{R_1\}$, where R_1 is the relation of the original $(R_1, \boxdot) \in \mathcal{C}$. Instead, if $s(p) = \mathtt{F}$ and $s(q) = \mathtt{U}$ we get $\mathcal{V}_s = \{PR, LR, P?, BC, S\}$ and $\mathcal{C}_s = \{R_1, R_2, R_3, R_5\}$, where this time R_1 is the intersection of the original $(R_1, \boxdot), (R_1, \neg p) \in \mathcal{C}$ since $s \models \boxdot$ and $s \models \neg p$.

Definition 6. A *schedule* for a subset of variables $\mathcal{V}' \subseteq \mathcal{V}$ is a mapping $\psi\colon \mathcal{V}' \to \bigcup_{V \in \mathcal{V}'} D(V)$ assigning values to the variables. We write Ψ to denote the set of all schedules. $\square$

Consider again Figure 3. $\psi(PR) = \mathtt{c}$ means that PR is assigned $\mathtt{c}$ (i.e., Charlie processes the loan request). At any time, the domain of ψ coincides with the set $\mathcal{V}_s$ of variables arising from the projection of the CNCU onto s (when s is initial, the domain of ψ consists of all unlabeled variables). However, a schedule is nothing but a fixed plan for executing a bunch of variables (not even saying in which order). The interesting part is how we generate it. To do so, we need a *strategy*. Let $\Delta(\mathcal{V}')$ be the set of all orderings for a subset $\mathcal{V}' \subseteq \mathcal{V}$, and $\Delta^* = \cup_i \{\Delta(\mathcal{V}_i)\}$ for any $\mathcal{V}_i \in 2^{\mathcal{V}}$ be the ordering universe.

Definition 7. An *execution strategy* for a CNCU $\mathcal{Z}$ is a mapping $\sigma\colon \mathcal{S} \to \Psi \times \Delta^*$ from scenarios to schedules and orderings such that the domain of the resulting schedule $\psi \in \Psi$ consists of all variables V belonging to the projection $\mathcal{Z}_s = \langle \mathcal{V}_s, \mathcal{D}, \mathcal{C}_s \rangle$ and $d \in \Delta^*$ is an ordering for $\mathcal{V}_s$. If $(\psi, d) = \sigma(s)$ also specifies a consistent assignment with d meeting the restriction $\prec$ of the initial CNCU, then σ is said to be *viable*. We write $val(\sigma, s, V) = v$ to denote the value v assigned to V by σ in s, and $ord(\sigma, s)$ to denote the ordering d assigned by σ in the scenario s to the variables in $\mathcal{V}_s$. $\square$

The first kind of controllability is *weak controllability* which ensures that each projection is consistent.

Definition 8. A CNCU is *weakly controllable* (WC) if for each complete scenario $s \in \mathcal{S}$ there exists a viable execution strategy $\sigma(s)$. $\square$

Dealing with such a controllability is quite complex as it always requires one to predict how all uncontrollable parts will behave before starting the execution. This leads us to consider the opposite case in which we want to synthesize a strategy working for all possible scenarios. Thus, the second kind of controllability is *strong controllability*.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

Definition 9. A CNCU is *strongly controllable* (SC) if there exists a viable execution strategy σ such that for any pair of honest scenarios s_1, s_2 and any variable V, if $V \in \mathcal{V}_{s_1} \cap \mathcal{V}_{s_2}$, then $val(\sigma, s_1, V) = val(\sigma, s_2, V)$ and $ord(\sigma, s_1) = ord(\sigma, s_2)$. $\square$

Strong controllability is, however, "too strong". If a CNCU is not strongly controllable, it could be still executable by refining the schedule in real time depending on how s evolves. To achieve this purpose, we introduce *dynamic controllability*. Since the truth values of propositions are revealed incrementally, we first introduce the formal definition of history that we then use to define dynamic controllability.

Definition 10. The *history* $\mathcal{H}(V,s)$ of a variable V in the scenario s is the set of all observations made before V executes. $\square$

Consider the projection of Figure 3 with respect to $s(p) = \mathtt{T}$ and $s(q) = \mathtt{U}$ and the ordering $d = PR \prec LR \prec P? \prec PC \prec Q? \prec S$. We have that $\mathcal{H}(PC,s) = \emptyset$ before $P?$ executes and $\mathcal{H}(PC,s) = \{p\}$ after.

Definition 11. A CNCU is *dynamically controllable* if there exists a viable execution strategy σ such that for any pair of scenarios s_1, s_2 and any variable $V \in \mathcal{V}_{s_1} \cap \mathcal{V}_{s_2}$ if $\mathcal{H}(V,s_1) = \mathcal{H}(V,s_2)$, then $val(\sigma,s_1,V) = val(\sigma,s_2,V)$ and $ord(\sigma,s_1) = ord(\sigma,s_2)$. $\square$

Abusing grammar, we use WC, SC and DC as both nouns and adjectives (the use will be clear from the context). As for temporal networks (Morris et al., 2001), it is easy to see that SC $\Rightarrow$ DC $\Rightarrow$ WC.

6 CONTROLLABILITY CHECKING ALGORITHMS

In this section, we provide the algorithms to check the three kinds of controllability introduced in Section 5. Since we are going to exploit directional consistency, we first need to address how to get a suitable total order for the variables meeting the restrictions specified by $\prec$. We will always classify as uncontrollable those CNCUs for which no total order exists.

Given a CNCU, to get a possible total order coherent with $\prec$, we build a directed graph G where the set of nodes is $\mathcal{V}$ and the set of edges is such that there exists a directed edge $V_1 \rightarrow V_2$ in G for any $(V_1,V_2) \in \prec$. We refer to this graph as $G = \langle \mathcal{V}, \prec \rangle$. For example, in Figure 3, G is the graph that remains after removing all labels and constraint edges.

From graph theory, we know that an ordering of the vertexes of a directed acyclic graph (DAG) meeting a given restriction $\prec$ can be found in polynomial time by running the TOPOLOGICALSORT algorithm

Algorithm 2: WC-CHECKING ($\mathcal{Z}$).

```
   Input: A CNCU $\mathcal{Z} = \langle \mathcal{V}, \mathcal{D}, D, \mathcal{OV}, \mathcal{P}, O, L, \prec, \mathcal{C} \rangle$
   Output: A set of solutions each one having the form
           $\langle s, d, Buckets \rangle$, where $s$ is a complete scenario, $d$ an
           ordering for $\mathcal{V}_s$ and $Buckets$ is a set of buckets (one for
           each variable in $\mathcal{V}_s$) if $\mathcal{Z}_s$ is WC, uncontrollable
           otherwise.
1  $Solutions \leftarrow \emptyset$
2  $HonestLabels \leftarrow$ COMPLETESCENARIOS($\mathcal{Z}$)
3  foreach $\ell \in HonestLabels$ do            ▷ For each honest scenario
4      Let $s$ be the scenario corresponding to $\ell$
5      Let $\mathcal{Z}_s$ be the projection of $\mathcal{Z}$ onto $s$
6      $d \leftarrow$ TOPOLOGICALSORT($G$)              ▷ $G \leftarrow \langle \mathcal{V}_s, \prec_s \rangle$
7      if no order is possible then
8          return uncontrollable
9      $Buckets \leftarrow$ ADC($\mathcal{Z}_s, d$)
10     if $\mathcal{Z}_s$ is inconsistent then
11         return uncontrollable
12     $Solutions \leftarrow Solutions \cup \{\langle s, d, Buckets \rangle\}$
13 return Solutions
```

Algorithm 3: CCCLOSURE($Labels$).

```
   Input: A set of labels Labels
   Output: The closure of all possible consistent conjunctions
1  $Closure \leftarrow Labels$
2  do
3      Pick two labels $\ell_1$ and $\ell_2$ from Closure
4      if $\ell_1 \wedge \ell_2$ is consistent and $\ell_1 \wedge \ell_2 \notin Closure$ then
5          $Closure \leftarrow Closure \cup \{\ell_1 \wedge \ell_2\}$
6  while Any adding is possible
7  return Closure
```

on G. At every step, TOPOLOGICALSORT chooses a vertex V without any predecessor (i.e., one without incoming edges), outputs V and removes V and all directed edges from V to any other vertex (equivalently, removes every $(V,V_2) \in \prec$). Then, TOPOLOGICALSORT recursively applies to the reduced graph until the set of vertexes becomes empty. If no total order exists, TOPOLOGICALSORT gets stuck in some iteration because of a cycle $V_1 \rightarrow \ldots V_1$, which makes impossible to find a vertex without any predecessor.

6.1 WC-checking

The idea behind the *weak controllability checking (WC-checking)* is quite simple: *every projection must have a solution.* Given a CNCU $\mathcal{Z} = \langle \mathcal{V}, \mathcal{D}, D, \mathcal{OV}, \mathcal{P}, O, L, \prec, \mathcal{C} \rangle$, we run the classic ADC on each projection $\mathcal{Z}_s$ according to a complete scenario s. Since each $\mathcal{Z}_s$ is a classic CN, any ordering (meeting the relevant part of $\prec$ for $\mathcal{Z}_s$) will be fine. We get one by running TOPOLOGICALSORT on $G_s = \langle \mathcal{V}_s, \prec_s \rangle$, where $\prec_s = \{(V_1,V_2) \mid (V_1,V_2) \in \prec \wedge V_1, V_2 \in \mathcal{V}_s\}$ (this is the relevant part of $\prec$). Since

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

Algorithm 4: COMPLETESCENARIOS($\mathcal{Z}$)

```
Input: A CNCU $\mathcal{Z} = \langle \mathcal{V}, \mathcal{D}, D, \mathcal{OV}, \mathcal{P}, O, L, \prec, \mathcal{C} \rangle$
Output: The set of all complete scenarios, where each $s$ is
        represented as the corresponding $\ell_s$.
1  $HonestLabels \leftarrow \{\boxdot\}$
2  for $P? \in \mathcal{OV}$ do
3      $HonestLabels \leftarrow$
         $HonestLabels \cup \{L(P?) \wedge p\} \cup \{L(P?) \wedge \neg p\}$
4  $HonestLabels \leftarrow$ CCCLOSURE($HonestLabels$)
5  do
6      Pick two labels $\ell_1$ and $\ell_2$ from $HonestLabels$
7      if $\ell_1 \neq \ell_2$ and $\ell_1 \Rightarrow \ell_2$ then
8          $HonestLabels \leftarrow HonestLabels \setminus \{\ell_2\}$
9  while Any removal is possible
10 return $HonestLabels$
```

there is no difference between honest labels and honest scenarios, we conveniently work with labels.

WC-CHECKING (Algorithm 2) starts by computing all complete scenarios (line 2). In a nutshell, it computes the longest consistent conjunctions arising from an initial set of labels containing (i) the empty label $\boxdot$, and (ii) for each observation variable $P?$, the pair of labels $L(P?) \wedge p$ (i.e., $L(P?)$ augmented with the positive literal p associated to $P?$), and $L(P?) \wedge \neg p$ (the other case) (Algorithm 4, lines 1-3). In this way, we consider all possible ways to *extend* an honest (but not necessarily complete) scenario. After that, Algorithm 4 computes the closure of all consistent conjunctions of labels drawn from this set (Algorithm 4, Algorithm 4, and more in detail Algorithm 3), and eventually rules out from the computed set of labels those entailed by some other different label in the same set. This is equivalent to saying that Algorithm 4 rules out all partial scenarios (last loop). In this way, we keep the longest conjunctions corresponding to all complete scenarios. Note that the conjunction of two honest and consistent labels is an honest and consistent label corresponding to a partial or a complete scenario.

The CNCU in Figure 3 is WC: the complete scenarios (written as labels) are $\neg p$, $p \wedge q$ and $p \wedge \neg q$.

For each complete scenario s, we synthesize a strategy σ by generating a solution for the projection $\mathcal{Z}_s$ following the ordering d computed initially (Algorithm 2, line 6). Although Definition 8 says that one strategy is enough, our approach is able to handle all possible strategies for each scenario s as during the solution-generation process the value assignments do not depend on any uncontrollable part.

As an example, consider Figure 3, the scenario $s_1 = \neg p$ and the ordering $d = PR \prec P? \prec LR \prec BC \prec S$ for $\mathcal{Z}_{\neg p}$. A possible strategy synthesized from the buckets of $\mathcal{Z}_{\neg p}$ along d and satisfying all constraints in Figure 4 is $val(\sigma, \neg p, PR) = \texttt{a}$, $val(\sigma, \neg p, P?) = \texttt{wf}$,

Algorithm 5: SC-CHECKING ($\mathcal{Z}$)

```
Input: A CNCU $\mathcal{Z} = \langle \mathcal{V}, \mathcal{D}, D, \mathcal{OV}, \mathcal{P}, O, L, \prec, \mathcal{C} \rangle$
Output: A tuple $\langle d, Buckets \rangle$, where $d$ is a total ordering for $\mathcal{V}$
        and $Buckets$ is a set of buckets (one for each variable) if
        $\mathcal{Z}$ is SC, uncontrollable otherwise.
1  Compute a CN $\mathcal{Z}_* \leftarrow \langle \mathcal{V}, \mathcal{D}, \mathcal{C}_* \rangle$ where $\mathcal{C}_* \leftarrow \{R_S \mid (R_S, \ell) \in \mathcal{C}\}$
2  $d \leftarrow$ TOPOLOGICALSORT($G$)              $\triangleright\ G \leftarrow \langle \mathcal{V}, \prec \rangle$
3  if no order is possible then
4      return uncontrollable
5  return ADC($\mathcal{Z}_*, d$)
```

$val(\sigma, \neg p, LR) = \texttt{c}$, $val(\sigma, \neg p, BC) = \texttt{d}$ and $val(\sigma, \neg p, S) = \texttt{b}$. That is, whenever we can predict that the workflow in Figure 2 is going through a business loan, Alice processes the request, Charlie logs it, David prepares the business contract and Bob does the signing. The relevant part of the complexity of WC-CHECKING is $2^{|\mathcal{P}|} \times Complexity(ADC)$ as the worst case is a CNCU specifying $2^{|\mathcal{P}|}$ complete scenarios (all other sub-algorithms run in polynomial time).

6.2 SC-checking

The *strong controllability checking (SC-checking)* does not need to unfold all honest scenarios at all. From an algorithmic point of view it is even easier to understand: *a single solution must work for all projections*. To achieve this purpose, we start with a simple operation: *we wipe out all the labels in the CNCU*. Then, we run ADC on this "super-projection" by choosing the ordering obtained by TOPOLOGICALSORT run on the related G (Algorithm 5).

The CNCU in Figure 3 is *not* SC. Although a total order exists once we have wiped out all the labels ($d = PR \prec P? \prec PC \prec Q? \prec LR \prec BC \prec S$), there is no way to find a consistent assignment to S that always works for the initial CNCU. It is not difficult to see that the problem lies in the constraints of the original CNCU shown in Figure 4. In the first phase, when ADC fills the buckets, each original constraint (R_S, ℓ) is deprived of its label ℓ (becoming $(R_S, \boxdot)$) and added to the bucket of the latest variable in S.

Consider the original $(R_6, p \wedge q)$ and $(R_6, p \wedge \neg q)$ (Figure 4). ADC transforms them into (two) *unlabeled* constraints $(R_6, \boxdot)$ and then add both to $Bucket(S)$. Since the labels of the two relations are the same, $Bucket(S)$ actually contains the intersection of the two (as both must hold). However, $(\{(\texttt{a},\texttt{b})\}, \boxdot) \cap (\{(\texttt{b},\texttt{b})\}, \boxdot) = (\emptyset, \boxdot)$.

In other words, in the workflow in Figure 2, Bob always does the signing. The problem is that the user who prepares the personal contract must be different according to which truth value `rnd?` will be assigned. If the system calls for a SoD (`rnd?` $= \texttt{T}$), then Alice

Algorithm 6: LABELEDADC$(\mathcal{Z}, d)$.

```
Input: A CNCU $\mathcal{Z} = \langle \mathcal{V}, \mathcal{D}, D, \mathcal{OV}, \mathcal{P}, O, L, \prec, \mathcal{C} \rangle$ and an
       ordering $d = V_1 \prec \cdots \prec V_n$
Output: A set Buckets of buckets (one for each variable) if $\mathcal{Z}$ is
        consistent along $d$, inconsistent otherwise.
1  foreach $(R_S, \ell) \in \mathcal{C}$ do                ▷ Partition constraints as follows
2      Let $V$ be the latest variable in $S$ according to $d$
3      Let $\ell_{Rem}$ be the conjunction of all literals $p$ or $\neg p$ in $\ell$ such
         that either $V = P?$ or $V \prec P?$ in $d$, where $P? = O(p)$
4      Add $(R_S, \ell - \ell_{Rem})$ to $Bucket(V)$
5  foreach $V$ in $d$ taken in reverse order do          ▷ Process buckets
6      $Closure \leftarrow$ CCCLOSURE$(\{\ell \mid (R_S, \ell) \in Bucket(V)\})$
7      for $\ell_n \in Closure$ do                      ▷ new constraint's label
8          $Entailed \leftarrow \{R_S \mid (R_S, \ell) \in Bucket(V) \wedge \ell_n \Rightarrow \ell\}$
9          $S_n \leftarrow \bigcup_{R_S \in Entailed} S \setminus \{V\}$       ▷ new constraint's scope
10         Compute $R_{tmp} \leftarrow \bowtie_{R_S \in Entailed} R_S$              ▷ enforce
             $k$-consistency
11         if $R_{tmp} = \emptyset$ then
12             return uncontrollable
13         if $S_n \neq \emptyset$ then              ▷ propagate the new constraint
14             $R_n \leftarrow \pi_{S_n}(R_{tmp})$       ▷ Project onto the new scope
15             Let $V_n$ be the latest variable in $S_n$ according to $d$
16             Compute $\ell_{Rem}$ as before but w.r.t. $\ell_n$
17             Add $(R_{S_n}, \ell_n - \ell_{Rem})$ to $Bucket(V_n)$
18 $Buckets \leftarrow \{\{Bucket(V)\} \mid V \in \mathcal{V}\}$
19 return Buckets
```

prepares the contract, else Bob does it. However, the intersection of the users allowed to carry out this task according to `rnd?` is empty, which means that the user who prepares the contract for a personal loan *cannot* be decided before the execution starts. The complexity of SC-CHECKING coincides with that of ADC as the sub-procedures to turn a CNCU unconditional and computing a total ordering run in polynomial time.

6.3 DC-checking

The *dynamic controllability checking (DC-checking)* addresses the most appealing type of controllability. If a CNCU is not SC, it could be DC by deciding which value to assign to which variable depending on how the uncontrollable part behaves. This subsection discusses this algorithm. We start with LABELEDADC (Algorithm 6), a main subalgorithm we make use of, which extends ADC to address the conditional part refining the adding or tightening of constraints to the buckets and the constraint-propagation.

When we add a constraint (R_S, ℓ) to a $Bucket(V)$, we *lighten* ℓ by removing all literals p or $\neg p$ in ℓ that will still be *unknown* by the time V executes. That is, those whose related observation variables are either V itself or will be assigned after V according to d.

When propagating constraints, LABELEDADC enforces the adequate level of k-consistency for all combinations of relevant honest (partial) scenarios arising from the conjunctions of all labels related to the constraints in the buckets. That is, for each V, it runs CCCLOSURE on the set $Closure = \{\ell \mid (R_S, \ell) \in Bucket(V)\}$. After that, it generates a new constraint (R_{S_n}, ℓ_n) for each $\ell_n \in Closure$, where S_n is the union of the scopes of the constraints in $Bucket(V)$ (whose labels are entailed by ℓ_n) deprived of V. R_{S_n} contains all tuples surviving the join of the entailed constraints projected onto S_n (as in the classic ADC). If no empty relation is computed, then the new constraint is added to the bucket of the latest variable in S_n (if any). If $S_n = \emptyset$, then it means that the algorithm computed an (implicit) unary constraint for V.

Finally, LABELEDADC returns the set of buckets from which any solution can be built *according to d*.

However, given an ordering d, if LABELEDADC "says no", it could be a matter of wrong ordering. Consider PC, $Q?$ and S in Figure 3, and assume that those three variables are ordered as $PC \prec Q? \prec S$. Further, consider Figure 4(g) and Figure 4(h), and suppose that $PC = \mathtt{a}$. When $Q?$ is executed ($Q? = \mathtt{wf}$), the truth value of q becomes known (recall that in this partial scenario $s(p) = \mathtt{T}$). If $s(q) = \mathtt{T}$, then $S = \mathtt{b}$ and $(\mathtt{a}, \mathtt{b}) \in (R_6, p \wedge q)$ (Figure 4(g)), but if $s(q) = \mathtt{F}$, then $S = \mathtt{b}$ and $(\mathtt{a}, \mathtt{b}) \notin (R_6, p \wedge \neg q)$ (Figure 4(h)).

More simply, if Alice executes `PContract` and afterwards `rnd?` = `F`, then no valid user remains for `Sign` as `SP`$_6$ calls for a BoD between the two tasks (Figure 1). If Bob executes `PContract` and afterwards `rnd?` = `T`, then the problem is the same (so there is no user who can be assigned conservatively to `PContract` without any information on `rnd?`). Fortunately, `PContract` and this split connector are *unordered* (no precedence is specified between the two components). This situation allows us to act in a more clever way: What if we executed `PContract` *after* observing the truth value of `rnd?`? In such a case, our *strategy* would be: if `rnd?` = `T`, then Alice, else Bob.

Formally, DC-CHECKING (Algorithm 7) works by looking for an ordering d coherent with $\prec$ such that LABELEDADC "says yes" when analyzing $\mathcal{Z}$ along d. If no ordering works, then the network is uncontrollable. The algorithm refines a recursive backtracking extension of *all topological sorts* to run LABELEDADC for every possible ordering meeting $\prec$. This makes DC-CHECKING incremental.

For example, the CNCU in Figure 3 is DC along the ordering $d_1 = PR \prec LR \prec P? \prec Q? \prec BC \prec PC \prec S$ and uncontrollable along $d_2 = PR \prec LR \prec P? \prec BC \prec PC \prec Q? \prec S$ (as PC is assigned before $Q?$).

We execute a CNCU proved to be DC as follows. Let ℓ_s be the label corresponding to the current scenario. Initially $\ell_s = \boxdot$. For each variable V along the

Algorithm 7: DC-CHECKING ($\mathcal{Z}$).

```
    Input: A CNCU $\mathcal{Z} = \langle \mathcal{V}, \mathcal{D}, D, \mathcal{OV}, \mathcal{P}, O, L, \prec, \mathcal{C} \rangle$
    Output: A tuple $\langle d, Buckets \rangle$, where $d$ is an ordering for $\mathcal{V}$ and
            $Buckets$ is a set of buckets (one for each variable) if $\mathcal{Z}$ is
            DC along $d$, uncontrollable otherwise.
1   $S \leftarrow \mathcal{V}$                                    ▷ global variable
2   $Buckets \leftarrow \emptyset$                                ▷ global variable
3   Let $d$ be an empty list for the ordering                    ▷ global variable
4   if ALLTOPSORTDC($S$) = true then
5       return $\langle d, Buckets \rangle$                       ▷ dynamically controllable
6   return uncontrollable
7   Procedure ALLTOPSORTDC($S$)                  ▷ $S$ is the current set of variables
8       if $S = \emptyset$ then                                  ▷ try the current order
9           $Buckets \leftarrow$ LABELEDADC($\mathcal{Z}, d$)
10          if $\mathcal{Z}$ is consistent then
11              return true                                     ▷ stop here
12      else
13          Let $\Pi$ be the set of variables without predecessors in $\prec$
14          for $V \in \Pi$ do
15              Add $V$ to the order $d$ as the last element
16              $S \leftarrow S \setminus \{V\}$
17              if ALLTOPSORTDC($S$) = true then
18                  return true
19              Remove the last element of $d$                   ▷ backtracking
20              $S \leftarrow S \cup \{V\}$                       ▷ backtracking
21      return false                            ▷ the ordering $d$ does not work
```

ordering d, if V is relevant for ℓ_s, then we look for a value v in the domain of V satisfying all relevant constraints in $Bucket(V)$. If V is irrelevant (as ℓ_s falsifies $L(V)$), then we ignore V and go ahead with the next variable (if any). Moreover, if V is an observation variable, where p is the associated proposition, then ℓ_s extends to $\ell_s \wedge p$ iff p is assigned `T`, and to $\ell_s \wedge \neg p$ otherwise. In this way, a partial scenario extends to a complete one, one observation variable at a time.

A strategy to execute the CNCU in Figure 3 is a strategy for the workflow in Figure 2: Charlie executes `ProcReq`, Alice `LogReq` and the workflow engine executes the first conditional split connector (always). If `pers?` = `F`, then David executes `BContract`. If `pers?` = `T`, then the workflow engine executes the second split connector to have full information on `rnd?`. If `rnd?` = `T`, then Alice executes `PContract`, else Bob. Bob executes `Sign` (always).

The complexity of DC-CHECKING is $\mathcal{V}! \times$ *Complexity*(LABELEDADC) as in the worst case there are $\mathcal{V}!$ orderings. We leave the investigation of the complexity of LABELEDADC as future work.

7 ZETA: A TOOL FOR CNCUS

We have developed ZETA, a tool for CNCUs that takes in input a specification of a CNCU and acts both as a solver for WC, SC and DC and as an executor.[1]

[1]ZETA is available at http://regis.di.univr.it/ICAART2018.tar.bz2 along with the set of benchmarks.

Given a CNCU specification file `network.cncu`, WC is checked by running `java -jar zeta.jar network.cncu --WCchecking network.ob` (we use `--SCchecking` and `--DCchecking` for SC and DC). If the CNCU is proved controllable, ZETA saves to file the *order and buckets* needed to later generate any solution (for WC, ZETA does so for any complete scenario). A controllable CNCU is executed by running `java -jar zeta.jar network.cncu --execute network.ob [N]`, where `[N]` (default 1) is the number of simulations we want to carry out. For WC, ZETA executes the CNCU with respect to each complete scenario, whereas for SC and DC, it executes the CNCU generating a random scenario (that is why ZETA allows for multiple simulations).

We ran ZETA on the CNCU in Figure 3. We used a FreeBSD virtual machine run on top of a VMWare ESXi Hypervisor using a physical machine equipped with an Intel i7 2.80GHz and 20GB of RAM. The VM was assigned 16GB of RAM and full CPU power. ZETA proved in about 200 milliseconds that the CNCU in Figure 3 is WC (saving an `ob`-file of 12Kb), is not SC but is DC (saving an `ob`-file of 8Kb). For WC and DC, the CNCU was correctly executed.

We implemented ZETA also in order to be able to carry out an automated and extensive experimental evaluation to compare the performances of the algorithms checking WC, SC and DC. We summarize our findings in the following.

We randomly generated an initial set of benchmarks of 10000 well-defined CNCUs. Each CNCU has 5 to 15 variables (of which minimum 1 and maximum 10 are observation variables), and 1 to 5 domains, where each domain is filled by sampling from the same initial random set of elements (10 to 30). Each CNCU is such that: each proposition labels some component, each domain is associated to at least one variable and a TOPOLOGICALSORT coherent with $\prec$ is possible. The number of constraints and tuples contained in them were generated to avoid underconstrained and overconstrained networks.

We ran WC, SC and DC-checking on this set imposing a time out of 300 seconds for each CNCU. Figure 5 shows the results, where the x-axis represents the # of analyzed instances, and the y-axis the overall time elapsed. ZETA first carried out the analysis for WC on the whole set of benchmarks proving that 3806 CNCUs were WC, 6189 were not WC and 5 hit the timeout. Then, it ran the analysis for SC proving that 3270 CNCUs were SC, 6730 were not SC and 0 hit the timeout. Finally, it ran the analysis for DC, proving that 3627 CNCUs were DC, 3034 were not DC and 3339 hit the timeout. We then confirmed, considering the CNCUs for which ZETA terminated

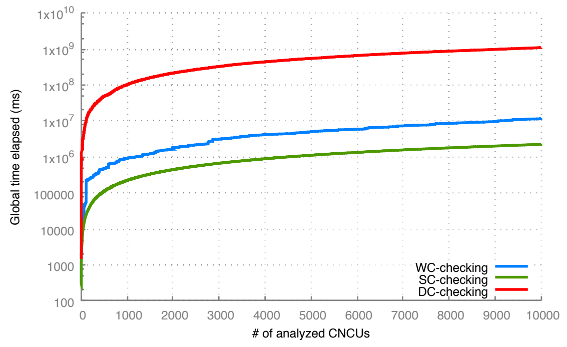


Figure 5: The race for controllability checking. SC (green, below) got 1st place as SC tests unconditional CNs. WC (blue, middle) got 2nd place as it tries all complete scenarios but not with respect to all possible orderings. DC (red, above) got 3rd place as it also looks for a suitable ordering.

within the timeout, that SC $\Rightarrow$ DC $\Rightarrow$ WC. Furthermore, 538 CNCUs were proved WC but not SC, 61 WC but not DC, and 357 DC but not SC. Finally, we executed 1000 times each controllable CNCU, shuffling the domains of the variables at every execution in order to get different solutions (if any). CNCUs proved WC were executed 1000 times with respect to each complete scenario. No execution crashed.

8 CORRECTNESS

Definition 12. A controllability algorithm is *sound* if, whenever it classifies a CNCUs as uncontrollable, the CNCU is really uncontrollable ($\Rightarrow$), and it is *complete* if, whenever a CNCU is uncontrollable, the algorithm classifies it as uncontrollable ($\Leftarrow$). $\square$

WC-CHECKING runs ADC on each projection $\mathcal{Z}_s$ corresponding to a complete scenario s. If ADC computes an empty relation, the CNCU is uncontrollable as there is no way to satisfy the constraints if s happens. Thus, WC-CHECKING is sound. WC-CHECKING is also complete because it does so for all complete scenarios guaranteeing that if all projections are consistent, then there exists a (possibly different) strategy for each complete scenario.

SC-CHECKING first wipes out the conditional part the original CNCU obtaining a super-projection whose set of constraints corresponds to the intersection of all sets of constraints (even inconsistent one another) related to all possible projections. Then it runs ADC on the resulting network. If ADC computes an empty relation, then it means that there is no way to decide some variable assignment before starting. Thus, SC-CHECKING is sound and complete as it boils down to the classic ADC.

Note that both WC-CHECKING and SC-CHECKING carry out the analysis on (possibly many) *unconditional* CNs. We point out that the chosen ordering according to $\prec$ given in input to ADC never breaches soundness and completeness of ADC but might only affect its complexity (Dechter, 2003).

LABELEDADC extends ADC to accommodate the propagation of labeled constraints. When it adds a constraint to the bucket of a variable V it lightens the label of the constraint by removing all literals whose truth value will be still unknown by the time V executes. This is because the observation variables associated to the propositions embedded in those literals will be executed *after* V or coincide with V itself. For this reason, we must be conservative and consider the constraint as if it just held, since we are unable to predict "what is going to be". LABELEDADC propagates the constraints enforcing the adequate level of k-consistency for all possible combinations of honest (partial) scenarios arising from the labels of the constraints in a bucket. If LABELEDADC detects an inconsistency, it means that there exists a (partial) scenario for which the value assignments to the variables of the CNCU (along the ordering in input) will violate some constraint. Thus, DC-CHECKING is sound as it runs LABELEDADC for all possible orderings. We believe that DC-CHECKING is also complete but leave a formal proof for future work.

9 RELATED WORK

CNs (Dechter, 2003) do not address uncontrollable parts and are thus incomparable with CNCUs.

A Mixed CSP (Fargier et al., 1996) partitions the set of variables in controllable and uncontrollable. Fargier et al. provide a consistency algorithm assuming full observability of the uncontrollable part. CNCUs do not have this restriction.

DCSPs (Mittal and Falkenhainer, 1990) introduce activity constraints saying when variables are relevant depending on the values assigned to some other variables. No uncontrollable parts are specified.

Some probabilistic approaches (e.g., (Fargier and Lang, 1993)) attempted to find the most probable working solution to a CSP under probabilistic uncertainty. Instead, our work addresses exact algorithms.

In a Prioritized Fuzzy Constraint Satisfaction Problem (PFCSP) (e.g., (Luo et al., 2003)) a solution threshold states the overall satisfaction degree. CNCUs do not deal with satisfaction degrees yet.

STNs, CSTNs, STNUs, CSTNUs and CSTNUDs only model temporal plans and are unable to represent resources. ACTNs (Combi et al., 2017) extend CST-

NUs to represent a dynamic user assignment that also depends on temporal aspects. CNCUs do not address temporal constraints for the good reason that directional consistency (CNs) allows for convergence when generating a solution only if a total ordering is followed. Most temporal networks do not have this restriction. ACTNs solve this problem by synthesizing memoryless execution strategies *before* starting.

WC, SC and DC are investigated for access-controlled workflows under conditional uncertainty in (Zavatteri et al., 2017). That work deals with structured workflows by unfolding workflow paths, considering binary constraints only (whose labels are the conjunction of the labels of the connected tasks) and assuming that a total order for the tasks is given in input. This work overcomes all these limitations.

10 CONCLUDING REMARKS

We introduced CNCUs to address a kind of CSP under conditional uncertainty. CNCUs implicitly embed classic CNs (if $\mathcal{OV} = \emptyset$ and $\prec = \emptyset$). We then defined and provided algorithms for WC, SC and DC. Currently, we only deal with CNCUs that are controllable with respect to a total ordering for the variables.

We discussed the correctness and complexity of our algorithms and provided ZETA, a tool for CNCUs that acts as a solver for WC, SC and DC as well as an execution simulator. We provided an extensive experimental evaluation against a set of benchmarks of 10000 CNCUs. SC is the easiest type of controllability to check, followed by WC and finally DC, which is currently the hardest one. DC is a matter of order (CNCUs not admitting any are uncontrollable). SC and DC provide usable strategies for executing workflows under conditional uncertainty. WC calls for predicting the future. However, WC is important because a CNCU proved non WC will never be SC nor DC.

As future work, we plan to work on the *all topological sort* phase of DC-CHECKING in order to contain the explosion of this step. We also plan to investigate if CNCUs classified as non-DC with respect to all possible total orderings might turn DC for some ordering that refines dynamically during execution.

REFERENCES

Cimatti, A., Hunsberger, L., Micheli, A., Posenato, R., and Roveri, M. (2016). Dynamic controllability via timed game automata. *Acta Inf.*, 53(6-8).

Cimatti, A., Micheli, A., and Roveri, M. (2015a). An SMT-based approach to weak controllability for disjunctive temporal problems with uncertainty. *Artif. Intell.*, 224.

Cimatti, A., Micheli, A., and Roveri, M. (2015b). Solving strong controllability of temporal problems with uncertainty using SMT. *Constraints*, 20(1).

Combi, C., Posenato, R., Viganò, L., and Zavatteri, M. (2017). Access controlled temporal networks. In *ICAART 2017*. INSTICC, ScitePress.

Dechter, R. (2003). *Constraint processing*. Elsevier.

Dechter, R., Meiri, I., and Pearl, J. (1991). Temporal constraint networks. *Artif. Intell.*, 49(1-3).

Dechter, R. and Pearl, J. (1987). Network-based heuristics for constraint-satisfaction problems. *Artif. Int.*, 34(1).

Fargier, H. and Lang, J. (1993). Uncertainty in constraint satisfaction problems: A probabilistic approach. In *ECSQARU '93*. Springer.

Fargier, H., Lang, J., and Schiex, T. (1996). Mixed constraint satisfaction: A framework for decision problems under incomplete knowledge. In *IAAI 96*.

Freuder, E. C. (1982). A sufficient condition for backtrack-free search. *J. ACM*, 29.

Gottlob, G. (2012). On minimal constraint networks. *Artif. Intell.*, 191-192.

Hunsberger, L., Posenato, R., and Combi, C. (2012). The Dynamic Controllability of Conditional STNs with Uncertainty. In *PlanEx 2012*.

Hunsberger, L., Posenato, R., and Combi, C. (2015). A sound-and-complete propagation-based algorithm for checking the dynamic consistency of conditional simple temporal networks. In *TIME 2015*.

Luo, X., Lee, J. H.-m., Leung, H.-f., and Jennings, N. R. (2003). Prioritised fuzzy constraint satisfaction problems: Axioms, instantiation and validation. *Fuzzy Sets Syst.*, 136(2).

Mackworth, A. K. (1977). Consistency in networks of relations. *Artif. Intell.*, 8(1).

Mittal, S. and Falkenhainer, B. (1990). Dynamic constraint satisfaction problems. In *AAAI 90*.

Montanari, U. (1974). Networks of constraints: Fundamental properties and applications to picture processing. *Inf. Sci.*, 7.

Morris, P. H., Muscettola, N., and Vidal, T. (2001). Dynamic control of plans with temporal uncertainty. In *IJCAI 2001*.

Tsamardinos, I., Vidal, T., and Pollack, M. E. (2003). CTP: A new constraint-based formalism for conditional, temporal planning. *Constraints*, 8(4).

Zavatteri, M. (2017). Conditional simple temporal networks with uncertainty and decisions. In *TIME 2017*, LIPIcs.

Zavatteri, M., Combi, C., Posenato, R., and Viganò, L. (2017). Weak, strong and dynamic controllability of access-controlled workflows under conditional uncertainty. In *BPM 2017*.