Coco: Runtime Reasoning About Conflicting Commitments

Nirav Ajmeri, Jiaming Jiang, Rada Chirkova, Jon Doyle, Munindar P. Singh
North Carolina State University, Raleigh, NC 27695, USA
{najmeri, jjiang13, rychirko, Jon_Doyle, singh}@ncsu.edu

Abstract

To interact effectively, agents must enter into commitments. What should an agent do when these commitments conflict? We describe Coco, an approach for reasoning about which specific commitments apply to specific parties in light of general types of commitments, specific circumstances, and dominance relations among specific commitments. Coco adapts answer-set programming to identify a maximal set of nondominated commitments. It provides a modeling language and tool geared to support practical applications.

1 Introduction

Major applications of information technology presuppose an ability to reason about interactions among autonomous parties. We characterize the expectations that parties have of one another using a notion of *commitment* in which general commitment *schemata* produce specific commitment *instances* when applied to specific parties in specific situations. In some situations, commitment instances demand conflicting conditions or assignments of resources. We characterize the means for resolving such conflicts in terms of situation-dependent *dominance* relations among commitments.

Example 1. *Alice, a child, takes ill. Society expects Alice's guardian, Bob, to take her immediately to her pediatrician, Carol, provided Alice requires immediate medical attention. Meanwhile, Bob's employer, Steve, expects him to work during business hours. Bob disregards his employer's expectation in light of the medical emergency involving his ward.* ■

Following Singh [2013], we use the term *principal* to mean an autonomous party (person, agent, organization, or polity) that can commit to others and be committed to by others. Example 1 includes as principals the people named and Society.

Bob is committed to Steve to stay at work during work hours, and to Society to take Alice to Carol if Alice becomes sick. If someday Alice falls sick during work hours, instances of the above schemata arising on that day conflict because Bob cannot be at two places at once. If that is all we know, there is no clear resolution. But the resolution is clear if it is more important for Bob to bring Alice to Carol when she is sick than to go to work.

Problem. *The problem we address is how to resolve commitment conflicts that are resolvable through runtime reasoning about situation-dependent dominance among instances.*

Novelty. Commitments, norms, and compliance have been extensively studied. Yet, despite many advances, previous work has not tackled the stated problem. Most treatments of commitments and norms do not represent dominance. Yolum and Singh [2004] and Alberti et al. [2006] compute feasible execution paths for agents or verify compliance. Dignum et al. [2005] define a CTL variant and present a formalism of deadlines to determine norm compliance based on whether agents bring about specified states before the deadline. Alechina et al. [2013] (hereinafter, ADL) apply CTL and ATL to reason about normative update. They incorporate deadlines and determine compliance via a notion similar to Dignum et al.'s. ADL's framework sanctions violators. Although Dignum et al. and ADL consider time, they do not consider dominance between norms. Knobbout and Dastani [2012] propose a logical framework by extending ATL to reason about agents' abilities under norm-compliance assumptions, but lack a notion of dominance.

Vasconcelos et al. [2009] and Governatori [2007] address conflicts between norms as a design problem with a view to specifying schemata (in our terminology) that avoid potential conflicts. In contrast, Coco aims to resolve conflicts as they arise between instances. This has the benefit of only having to deal with conflicts that occur, not those that might occur. The contrary-to-duty literature, e.g., [Prakken and Sergot, 1996], deals with secondary obligations that come into effect when the primary obligations are violated.

Boella and van der Torre [2008] provide a mechanism for expressing which norm supersedes which other in which situation, but their work does not address automatic runtime identification of nonsuperceded instances.

Some heuristic approaches are promising. Oren et al. [2008] propose an argumentation-based heuristic to determine which norms to comply with when norms conflict. They build conflict graphs with norms and conflicts as nodes and edges. Oren et al. use agent preferences to prune norms in conflict graphs, and permanently drop nonpreferred norms. Oren et al.'s work ignores temporal aspects. In contrast, Coco deals with commitments at runtime.

Criado et al. [2015] propose an economics-based approach

that resolves conflicts among instances of norms by optimization using agent objectives and preferences. The optimization function, built into the conflict-resolution algorithm, considers the coherence of the available cognitive and normative elements. In contrast, Coco uses explicit rules to specify dominance among commitment instances. These rules enable tracing and interpretation of conflict resolutions in terms of agent objectives and preferences.

Contributions. Coco provides representations for commitments and dominance, as well as tractable decision procedures for determining compliance of actions with commitment instances. Coco (1) applies answer-set programming (ASP) [Gelfond, 2008; Gebser *et al.*, 2014] to identify nondominated commitments given knowledge of the world and of the specific situation, and (2) applies ADL's techniques to determine compliance of agent actions with nondominated commitment instances. This paper also provides theoretical results on the correctness and tractability of the problems of (3) determining maximal sets of nondominated commitment instances, and (4) determining liveness and safety of a system of commitments.

2 Specifying Commitments

A commitment here involves two principals, with the *creditor* principal committed in some way to the *debtor* principal. A commitment schema $(dbt, crd, \phi, \psi, t_d)$ indicates that debtor dbt is committed to creditor crd so that if the *antecedent* condition ϕ holds, then the *consequent* condition ψ should hold at some point within the *deadline duration* t_d, meaning that ψ should hold at some time within time units after ϕ begins to hold.

We write $(s, dbtI, crdI, t_i)$ to mean a commitment instance in which s is the commitment schema that the instance instantiates, $dbtI$ and $crdI$ are the specific debtor and creditor (i.e., the principals who fill the corresponding roles in the schema), and t_i is the time at which the instance becomes *detached*, i.e., at which the schema's antecedent for the particular debtor and creditor becomes true.

Listing 1 formalizes a commitment schema, *gCom*, committing guardians to take care of their charges, as open (implicitly universally quantified) FOL formulas. We use capitalized words to represent variables, and lowercase for constants. Statements 2 and 3 mean that if G fills the guardian role for charge C at T, then G is the debtor of the *gCom* instance, and C is the creditor. Statements 4 and 5 state the antecedent and consequent, respectively. The antecedent is that C falls sick, and the consequent is that G brings C to C's pediatrician Ped. Line 6 states that the deadline duration is 3 time units.

Similarly, the schema *wCom* defined in Listing 2 commits employees to be at work during work hours. If employee Ee's employer is Er at time T, then Ee is the debtor and Er is the creditor of the *wCom* instance. The antecedent of *wCom* is that time T is a work hour for employee Ee at employer Er, and the consequent is Ee is at work for Er at time T. The deadline duration is 0 time units.

```
1  schema(gCom)
2  guardianR(C, G, T) → dbt(gCom, G, T)
3  guardianR(C, G, T) → crd(gCom, C, T)
4  [sick(C, T) ∧ crd(gCom, C, T)] → ant(gCom, T)
5  [bring(G, C, Ped, T) ∧ pedR(C, Ped, T)
       ∧dbt(gCom, G, T) ∧ crd(gCom, C, T)]
       → con(gCom, T)
6  dDuration(gCom, 3)
```

Listing 1: Definition of the gCom schema

```
1  schema(wCom)
2  employer(Ee, Er, T) → dbt(wCom, Ee, T)
3  guardianR(C, G, T) → crd(wCom, Er, T)
4  [dbt(gCom, Ee, T) ∧ crd(wCom, Er, T)
       ∧workhour(Ee, Er, T)] → ant(wCom, T)
5  [dbt(wCom, Ee, T) ∧ crd(wCom, Er, T)
       ∧atWork(Ee, Er, T)] → con(wCom, T)
6  dDuration(wCom, 0)
```

Listing 2: Definition of the wCom schema

We treat compliance and conflict with respect to instances. Thus, if Alice falls sick at time instant T, an instance of *gCom* is instantiated and becomes detached at T. If Bob takes Alice to her pediatrician within three hours, then the instance is satisfied. Otherwise, the instance is violated.

3 Formal Language and Model

Coco's formal language is the function-free first-order predicate calculus with a distinguished set of predicates and constants. To describe facts of the world, it contains (1) constants naming all principals in lowercase, such as $alice$ and bob, distinguished times, such as time 10, within a finite interval, and durations; (2) predicates naming roles of organizations, such as $guardianR$, indicated by names ending in R; (3) predicates naming conditions of interest, such as $workhour$; (4) predicates naming actions, such as $bring$; and (5) predicates $<$ (precedes), $=$ (equals), and $\leq$ (precedes or equals) for comparing times.

To describe commitments, the language contains (1) constants naming schemata; and (2) terms naming instances of schemata. For example, the ground term $instance(gCom, bob, alice, 10)$, which we abbreviate to $gComInst$ in the following, names an instance of $gCom$ with debtor bob and creditor $alice$ detached at time 10. Similarly, the term $instance(wCom, bob, steve, 11)$, abbreviated henceforth as $wComInst$, names an instance of $wCom$.

The methods described in Section 4.2 for testing liveness and safety require a function-free language. When forming instances, Coco asserts statements that identify the components of an instance term, as illustrated for $gComInst$ in Listing 3.

```
1  [sInst(gComInst)∧isInstOf(gComInst, gCom)]
       → crdI(gComInst, alice)
2  [sInst(gComInst)∧isInstOf(gComInst, gCom)]
       → dbtI(gComInst, bob)
3  dDurationI(gComInst, 3)
```

Listing 3: Some properties of gComInst

The antecedent and consequent of an instance, omitted in the listing, are formed by substituting the names of specific debtor and creditor for the corresponding variables in the schema.

Table 1 lists the main predicates of schemata and instances of Coco. S is a refers to a schema, and Si refers to an instance. $schema$ and $sInst$ specify the corresponding types; $isInstOf$ identifies the schema of an instance; and $antI$ and $conI$ state that instance antecedents and consequents hold at specific times.

Schema predicates	
$schema(S)$	$ant(S,T)$
$dbt(S,P,T)$	$con(S,T)$
$crd(S,P,T)$	$dDuration(S,T)$
Instance predicates	
$sInst(Si)$	$violated(Si,T)$
$isInstOf(Si,S)$	$satisfied(Si,T)$
$dbtI(Si,P)$	$becomesDetached(Si,T)$
$crdI(Si,P)$	$becomesViolated(Si,T)$
$antI(Si,T)$	$becomesSat(Si,T)$
$conI(Si,T)$	$conflicting(Si_1,Si_2,T)$
$dDurationI(Si,T)$	$dominates(Si_1,Si_2,T)$
$detached(Si,T)$	$dominated(Si,T)$
$sameDbtI$	$conINotHold(Si,T_1,T)$
$sameCrdI$	

Table 1: Main predicates of schemata and instances.

The satisfaction of an instance is formalized as $becomesSat(Si,T)$, which means that T is the first time at which the consequent of Si holds, provided this happens before the deadline; *satisfied* is true of the instance from that time on. The violation of an instance is realized in a similar way. Specifically, the statement $becomesViolated(Si,T)$ holds at T when Si is detached, but not dominated, and its consequent does not hold within the deadline. The definitions for *becomesSat* and *becomesViolated* are provided in Listing 4. The predicate $conINotHold(Si,T_1,T)$ says that the consequent of Si is not true between times T_1 and T. The two utility predicates $difference(T_1,T_2,T)$ and $addition(T_1,T_2,T')$ state that the difference and sum of T_1 and T_2 are T and T', respectively.

1 $[detached(Si,T') \wedge conI(Si,T) \wedge difference(T,T',1)]$
$\rightarrow becomesSat(Si,T)$
2 $[becomesDetached(Si,T_1) \wedge dDurationI(Si,T_2)$
$\wedge addition(T_1,T_2,T) \wedge conINotHold(Si,T_1,T)$
$\wedge \neg dominated(Si,T)] \rightarrow becomesViolated(Si,T)$

Listing 4: Satisfaction and violation relations

We define a set of instances as conflicting if the set of their consequents cannot hold simultaneously. Whether a consequent can be true when another consequent is true depends on the domain knowledge of specific scenarios. Section 5 formalizes knowledge showing how *gComInst* and *wComInst* conflict in our example. Listing 5 defines $conflicting(Si_1,Si_2,T)$ symmetrically as holding at time T in case Si_1 and Si_2 are detached but their consequents cannot both hold at T.

1 $[detached(Si_1,T) \wedge detached(Si_2,T)$
$\wedge \neg(conI(Si_1,T) \leftrightarrow conI(Si_2,T))]$
$\rightarrow conflicting(Si_1,Si_2,T)$

Listing 5: The conflicting relation

In Example 1, Bob is required to be at his workplace during work hours and at Carol's office with Alice when Alice is sick. If Alice gets sick during Bob's work hours, both *gComInst* and *wComInst* detach and require Bob to be in two places at the same time. The consequent conditions conflict even when Alice is not sick, but only matter for detached instances; only then may such conflicts cause a commitment violation.

We resolve such conflicts by employing dominance relations, so that satisfaction of a more dominant instance vitiates violation of a less dominant instance. Thus, a principal can comply with one instance while violating another, and still remain in compliance with the entire set of instances. We require that the relation *dominates* (more dominant than) be a strict partial order, that is, transitive and irreflexive. A partial order is appropriate because changing commitments and unforeseen conflicts may make a total ordering infeasible; a partial order can suffice to resolve conflicts that do arise. We select from the set of all detached instances a maximal subset of *nondominated* instances, that is, a subset of detached instances none of which is less dominant than some conflicting instance in the same set.

Listing 6 formalizes simple dominance of *gCom* over *wCom*. The statement says that a detached instance of *gCom* dominates a detached instance of *wCom* if they have the same debtor, which is indicated by the utility predicate $sameDbtI$. Section 5.2 gives examples in which this dominance is narrowed by requiring additional conditions.

1 $[isInstOf(Si_1,gCom) \wedge isInstOf(Si_2,wCom)$
$\wedge detached(Si_1,T) \wedge detached(Si_2,T)$
$\wedge sameDbtI(Si_1,Si_2)] \rightarrow dominates(Si_1,Si_2,T)$

Listing 6: The guardian dominance relation

Listing 7 defines $dominated(Si_1,T)$ to hold at T in case a detached instance Si_1 is in conflict with another detached, nondominated instance Si_2 that dominates it at T.

1 $[dominates(Si_2,Si_1,T) \wedge conflicting(Si_1,Si_2,T)$
$\wedge \neg dominated(Si_2,T) \wedge detached(Si_1,T)$
$\wedge detached(Si_2,T)] \rightarrow dominated(Si_1,T)$

Listing 7: The dominated relation

4 Decision Procedures, Theoretical Results

Given a Coco snapshot of the current enactment, specifically the current state of the knowledge on the detached instances and the appropriate background knowledge, we test whether the snapshot admits an unambiguous nondominated set of currently detached instances, which we call a *nondominated set.* If it does, we obtain a maximal such set, $\mathcal{D}$, and then proceed to check the liveness and safety properties of $\mathcal{D}$.

In Ex. 1, the set $\mathcal{D}$ would contain (i) a nondominated instance for Bob taking Alice to the hospital, and (ii) a domi-

nated instance for Bob having to be at work at the same time.

4.1 Testing for Existence of Nondominated Set

We test enactments for admission of nondominated sets of detached instances by obtaining stable models under ASP semantics, e.g., [Gelfond, 2008], and by interpreting the outputs in our setting. Intuitively, we take a snapshot $\mathcal{S}$ as input, and return a *stable model* of $\mathcal{S}$ under ASP semantics, that is, all the facts implied by $\mathcal{S}$ under the semantics that do not become negated under inferences using the rules of $\mathcal{S}$.

Definition 1. *For a noncontradictory snapshot $\mathcal{S}$ of knowledge about an enactment, we call a set $\mathcal{D}$ of detached instances in $\mathcal{S}$ a* nondominated set of detached instances *whenever for each instance ni_i in $\mathcal{D}$, the following holds:*

(a) It is not possible to infer (under ASP) from $\mathcal{S}$ the fact dominates(ni_j, ni_i, t) for any detached instance ni_j in $\mathcal{S}$ and time t at which ni_i and ni_j are in conflict; and

(b) It is not possible to infer from $\mathcal{S}$ the fact that ni_i conflicts with any other detached instance in $\mathcal{D}$ at time t.

We call any such set $\mathcal{D}$ saturated *if it includes each detached instance in $\mathcal{S}$ that satisfies condition (a) above.*

The intuition for saturated sets of detached instances is that each such set witnesses that for each pair in $\mathcal{S}$ of detached instances ni_i and ni_j that are in conflict at time t, $\mathcal{S}$ entails the fact of dominance at time t of either ni_i by ni_j or vice versa. That is, our dominance knowledge is complete in this case for all the detached instances in snapshot $\mathcal{S}$.

Given a snapshot $\mathcal{S}$, with each stable model $\mathcal{M}$ of $\mathcal{S}$ we associate the *$\mathcal{M}$-nondominated set of detached instances in $\mathcal{S}$*, written $\mathcal{D}(\mathcal{S}, \mathcal{M})$, as follows: $\mathcal{D}(\mathcal{S}, \mathcal{M})$ is a maximal set of detached instances $ni_i \in \mathcal{S}$ such that

(1) $\mathcal{M}$ does not include the fact *dominates*(ni_j, ni_i, t) for any detached instance ni_j in $\mathcal{S}$ at time t, and

(2) It is not possible to infer from $\mathcal{S}$ the fact that ni_i conflicts with any other detached instance in $\mathcal{D}(\mathcal{S}, \mathcal{M})$.

The set $\mathcal{D}(\mathcal{S}, \mathcal{M})$ may or may not be saturated, depending on the degree of completeness of the dominance knowledge that can be inferred from the snapshot $\mathcal{S}$. (For example, if $\mathcal{S}$ entails the fact of conflict of two instances, ni_i and ni_j, but does not entail any dominance knowledge, then two sets $\mathcal{D}(\mathcal{S}, \mathcal{M}) = \{ni_i\}$ and $\mathcal{D}(\mathcal{S}, \mathcal{M}') = \{ni_j\}$ may exist, with neither set being saturated or saturable.)

We further say that two stable models $\mathcal{M}$ and $\mathcal{M}'$ of $\mathcal{S}$ are *nondominance identical* if and only if their associated sets $\mathcal{D}(\mathcal{S}, \mathcal{M})$ and $\mathcal{D}(\mathcal{S}, \mathcal{M}')$ are the same.

Theorem 1. *Given a noncontradictory snapshot $\mathcal{S}$: At least one stable model for $\mathcal{S}$ exists and is nondominance identical to all stable models for $\mathcal{S}$ if and only if there exists for $\mathcal{S}$ a saturated nondominated set of detached instances in $\mathcal{S}$.*

For instance, in Example 1 there exists a single stable model with instances (i) and (ii) as in the beginning of this section, provided $\mathcal{S}$ captures that *gCom* dominates *wCom* when Alice is sick. Otherwise, ASP would compute two or more stable models, an indication that $\mathcal{S}$ does not admit a saturated nondominated set of detached instances in $\mathcal{S}$.

Proof. (sketch) We begin by proving the existence claim of Theorem 1. The only if part is immediate from the definitions.

For the if part, suppose that there exists for $\mathcal{S}$ a saturated nondominated set, $\mathcal{D}$, of detached instances in $\mathcal{S}$. We show that in this case there exists at least one stable model, $\mathcal{M}$, for the snapshot $\mathcal{S}$, such that $\mathcal{D}$ is the $\mathcal{D}(\mathcal{S}, \mathcal{M})$ associated with $\mathcal{M}$. Indeed, by definition of $\mathcal{D}$, for each element ni_i of $\mathcal{D}$ one cannot infer from $\mathcal{S}$ that ni_i is in conflict with another detached instance that dominates it. Further, $\mathcal{D}$ includes all the instances in $\mathcal{S}$ that satisfy the condition (a) in Definition 1. Thus, $\mathcal{D}$ is a unique and maximal set w.r.t. the detached instances entailed from $\mathcal{S}$. We now obtain the set $\mathcal{J}$ of all the facts implied from $\mathcal{S}$; the set $\mathcal{M} = \mathcal{S} \cup \mathcal{J}$ is a stable model of $\mathcal{S}$ by construction. It is immediate from the definitions that $\mathcal{D}$ is the set $\mathcal{D}(\mathcal{S}, \mathcal{M})$.

We now prove by contradiction that under the assumptions of the previous paragraph, all the stable models of the given snapshot $\mathcal{S}$ are nondominance identical. Indeed, assume there exists a stable model, $\mathcal{M}'$, of the snapshot $\mathcal{S}$, such that the set $\mathcal{D}(\mathcal{S}, \mathcal{M}')$ is not identical to the saturated set $\mathcal{D}$ as defined above. By definition of $\mathcal{D}$, the only way this can happen is when $\mathcal{D}(\mathcal{S}, \mathcal{M}')$ is a proper subset of $\mathcal{D}$. (Recall that the snapshot $\mathcal{S}$ is noncontradictory.) This means that for at least one detached instance $ni_i \in \mathcal{S}$, such that $ni_i \notin \mathcal{D}(\mathcal{S}, \mathcal{M}')$, one cannot derive from $\mathcal{S}$ that ni_i is dominated by another detached instance $ni_j \in \mathcal{S}$, at time instant t. At the same time, ni_i is not in conflict with any element of $\mathcal{D}(\mathcal{S}, \mathcal{M}')$, as witnessed by the existence of the set $\mathcal{D}$. It follows that the set $\mathcal{D}(\mathcal{S}, \mathcal{M}')$ is not maximal as required by definition of associated set for $\mathcal{M}'$, a contradiction. □

Theorem 2 is immediate from Theorem 1.

Theorem 2. *Given a noncontradictory snapshot $\mathcal{S}$: The asymptotic runtime complexity of (a) determining whether a saturated nondominated set of detached instances in $\mathcal{S}$ exists and (in case it exists) of (b) producing such a set coincides with the runtime complexity of finding the stable models of $\mathcal{S}$.*

4.2 Liveness and Safety Checking

Suppose that for a snapshot $\mathcal{S}$, there exists a saturated nondominated set $\mathcal{D}$ of detached instances in $\mathcal{S}$. Then it follows from Theorems 1 and 2 that it is decidable and tractable to construct $\mathcal{D}$ using a standard ASP solver. Suppose that we further want to check whether certain liveness and safety properties hold in $\mathcal{S}$ in presence of $\mathcal{D}$.

Following ADL, we understand *liveness* as capturing that desirable states (from the designer's point of view) are possible w.r.t. the current enactment and can be achieved without an agent incurring sanctions, and *safety* as capturing that it is impossible for an agent to reach a state where some undesirable property holds without incurring some sanctions. Our approach for checking liveness and safety properties is by reduction to ADL's approach, as follows.

Given a liveness or safety property, φ, in the context of a snapshot $\mathcal{S}$ and of its nondominated set $\mathcal{D}$, we translate each of φ, $\mathcal{S}$, and $\mathcal{D}$ into propositions by using separate proposition names for all possible instantiations of the predicates in φ, $\mathcal{S}$, and $\mathcal{D}$ with the constants in the given universe of discourse. (We assume there is a constant upper bound on the arity of

the predicates and rules in $\mathcal{S}$, φ, and $\mathcal{D}$. That is, we do not consider the arity of the facts or rules in $\mathcal{S}$, φ, or $\mathcal{D}$ a variable part of the input.) We then treat the result of this process, $\mathcal{S}_2$, as an input to ADL's approach, and treat the output of the latter approach as follows.

Theorem 3. *Let $\mathcal{S}_2$ be a knowledge base as described above and including a suitable transformation of a liveness or safety property, φ, on a snapshot $\mathcal{S}$ that admits a saturated nondominated set $\mathcal{D}$. Then ADL's algorithm [2013, proof of their Theorem 1], when given $\mathcal{S}_2$ as its input, outputs the answer "yes" (respectively, "no") if and only if the property φ holds (respectively, does not hold) on $\mathcal{S}$ in presence of $\mathcal{D}$.*

Proof. ADL express liveness and safety properties in CTLS, a sanctions-based extension of CTL [Clarke *et al.*, 1986]. ADL check a liveness or safety property φ against a set N of conditional norms and a transition system M via model checking. (Due to the page limit, we refer the reader to Alechina et al. [2013] for the details.) Model checking takes as inputs a transition system M with initial state s, a finite set of conditional norms N, and a formula φ of CTLS. It returns true if M^N, $s \models \psi$, and returns false otherwise. Here, M^N is a *normative update* of M with N, with the meaning of an expansion of M into all possible complete paths (runs) and with all the norms in N enforced on all the runs.

It is straightforward to verify that (1) the transformation of $\mathcal{S}$, φ, and $\mathcal{D}$ into $\mathcal{S}_2$ described above results in $\mathcal{S}_2$ that is a correct "propositional expansion" of the inputs, and of size bounded by a polynomial in the size of the inputs, and that (2) the result $\mathcal{S}_2$ of the translation constitutes a valid input to ADL's checkers. As a result, ADL's checkers provide correct answers for $\mathcal{S}$, φ, and $\mathcal{D}$ via providing correct answers for $\mathcal{S}_2$ and via $\mathcal{S}_2$ being their correct "propositional expansion." □

By the results of Alechina et al. [2013], the model-checking problem for a normative update of a transition system is PSPACE complete; the proofs are constructive and thus provide a PSPACE-complete algorithm for verifying whether a given liveness or safety property holds against a set of conditional norms and a transition system.

Consequently, checking liveness and safety properties of enactments that admit saturated nondominated sets of instances is decidable with asymptotic runtime complexity of PSPACE, equal to that of ADL's approach.

5 Demonstration

We now demonstrate the Coco workflow and formalize conflict detection using Example 1. We then show how to add constraints to dominance relations in Section 5.2.

5.1 Formalizing the Example

Figure 1 illustrates Example 1, omitting time instants before 10 and after 13 for brevity. Each node represents the state update of an instance. One time instant contains multiple nodes if there are state updates of multiple instances at that time. The line above a node is the debtor's action, represented as a predicate, that caused the state update. For example, Alice becomes sick at time 10, which causes detachment of *gComInst*. Then at time 11, the middle node (labeled 11) indicates detachment of *wComInst*. Node 11b says *wComInst* becomes satisfied if Bob is at work at 11. Similarly, 11a says *gComInst* becomes satisfied if Bob brings Alice to her pediatrician at time 11. A conflict occurs at time 11 because Bob cannot be at two places at once, assuming Carol's office is at $hospitalLoc$, Bob's work is at $companyLoc$, and the two are distinct. Finally, node 13 indicates that with the conflict detection and dominance relation, *gComInst* would not become violated even if Bob did not bring about the consequent.

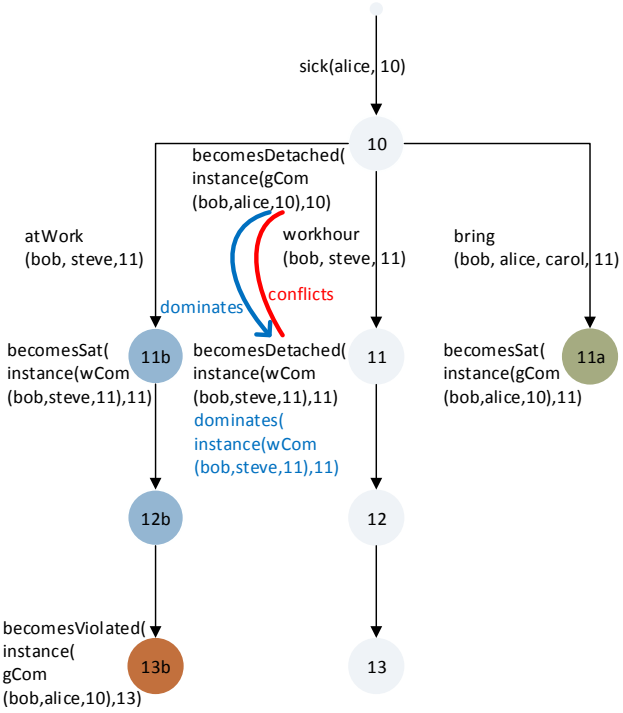

Figure 1: A healthcare scenario (see Example 1)

In summary, Coco finds that *gComInst* dominates *wComInst* at time 11; *gComInst* is satisfied at 11 (when Bob brings Alice to Carol). The execution demonstrates how, in the presence of dominance relations, an enactment can comply with a set of instances without complying with each of them.

Consider the scenario where, instead of bringing Alice to the hospital when she is sick, Bob chooses to work, illustrated in Figure 1 by the path 10→11b→12b→13b. Coco produces an output similar to the previous case but finds that *gComInst* is violated at time 13 and *wComInst* is satisfied at 11. The nondominated instances remain violated if their consequents do not hold within the deadline, whereas a dominated instance may be satisfied.

Listing 8 formalizes the conflict detection for this enactment. Statement 1 says the same principal cannot be at two different locations at the same time. Statements 2–4 imply that *wComInst*'s consequent cannot be true at T when *gComInst*'s consequent holds at T. Lines 5–7 handle the converse. Thus, *gComInst* and *wComInst* instances conflict and the latter is dominated.

1 $[atLocation(P, L_1, T) \wedge (L_1 \neq L_2)]$
$\rightarrow \neg atLocation(P, L_2, T)$

2 $[isInstOf(Si, gCom) \wedge conI(Si, T) \wedge dbtI(Si, G)]$
$\rightarrow atLocation(G, hospitalLoc, T)$

3 $[employerR(Ee, Er, T)$
$\wedge\neg atLocation(Ee, companyLoc, T)]$
$\rightarrow \neg atWork(Ee, Er, T)$

4 $[isInstOf(Si, wCom) \wedge dbtI(Si, Ee) \wedge crdI(Si, Er)$
$\wedge\neg atWork(Ee, Er, T)] \rightarrow \neg conI(Si, T)$

5 $[isInstOf(Si, wCom) \wedge conI(Si, T) \wedge dbtI(Si, Ee)$
$\wedge crdI(Si, Er)] \rightarrow atLocation(Ee, company, T)$

6 $[guardianR(C, G, T) \wedge pedR(C, Ped, T)$
$\wedge\neg atLocation(G, hospitalLoc, T)]$
$\rightarrow \neg bring(G, C, Ped, T)$

7 $[isInstOf(Si, gCom) \wedge dbtI(Si, G) \wedge crdI(Si, C)$
$\wedge pedR(C, Ped, T) \wedge \neg bring(G, C, Ped, T)]$
$\rightarrow \neg conI(Si, T)$

Listing 8: Conflict-detection knowledge for Example 1

5.2 Additional Dominance Relations

Dominance relations can be specified incrementally to cover conflicts between different types of commitments in different circumstances. For example, suppose Bob's wife, Amy, is also available and responsible for Alice, and consider the following modified scenario.

Example 2. *Alice, Bob and Amy are all at home and Bob needs to go to work. Alice suddenly takes ill and Bob would be late for work if he takes Alice to her pediatrician. Meanwhile, Amy is taking a day off and available for taking care of Alice. So Bob goes to work and Amy immediately takes Alice to the hospital.* ■

Formalizing Example 2 involves another instance of *gCom*, namely *instance(gCom, amy, alice, 10)*, abbreviated *gComInst2*. We want *gComInst2* to dominate *gComInst* because Amy is currently available but Bob is not. We formalize this availability-dependent dominance relation in Listing 9. Specifically, a detached instance Si_1 of *gCom* dominates another detached instance Si_2 of *gCom* if they have the same creditor, Alice in this case, but different debtors, Bob and Amy in this case, and Si_1's debtor is available while Si_2's is not.

1 $[isInstOf(Si_1, gCom) \wedge isInstOf(Si_2, gCom)$
$\wedge detached(Si_1, T) \wedge detached(Si_2, T)$
$\wedge sameCrdI(Si_1, Si_2) \wedge dbtI(Si_1, G_1)$
$\wedge dbtI(Si_2, G_2) \wedge (G_1 \neq G_2)$
$\wedge available(G_1, T) \wedge \neg available(G_2, T)]$
$\rightarrow dominates(Si_1, Si_2, T)$

Listing 9: The availability dominance relation

In Example 2, *gComInst2* is nondominated. However, *gComInst2* dominates and conflicts with *gComInst*, so *gComInst* is dominated. As before, *gComInst* dominates *wComInst*, so by transitivity, *gComInst2* dominates *wComInst*. Although *gComInst* still conflicts with *wComInst*, *wComInst* is nondominated because *gComInst* is dominated and because it does not conflict with *gComInst2*.

Consider the situation where neither Bob nor Amy is available, then we could augment the dominance relation given in Listing 9 to say that whoever is closer to Alice should take her to the hospital. Listing 10 formalizes this using the *dist(G,C,D,T)* predicate to say the distance between G and C at time T is D.

1 $[isInstOf(Si_1, gCom) \wedge isInstOf(Si_2, gCom)$
$\wedge detached(Si_1, T) \wedge detached(Si_2, T)$
$\wedge sameCrdI(Si_1, Si_2) \wedge crdI(Si_1, C)$
$\wedge dbtI(Si_1, G_1) \wedge dbtI(Si_2, G_2) \wedge (G_1 \neq G_2)$
$\wedge\neg available(G_1, T) \wedge \neg available(G_2, T)$
$\wedge dist(G_1, C, D_1, T) \wedge dist(G_2, C, D_2, T)$
$\wedge(D_1 \leq D_2)] \rightarrow dominates(Si_1, Si_2, T)$

Listing 10: Distance Dominance Relation

6 Discussion

Coco provides a flexible formalization of instances, including reasoning about conflicts and dominance. It employs ASP to compute maximal consistent sets of instances and adapts ADL's techniques to tackle compliance, liveness, and safety. Coco supports identifying and dealing with conflicts between commitments arising at runtime.

Additional related work. Besides the works discussed in Section 1, works on policy reasoning and software engineering are relevant. Marinovic et al. [2014] formalize "break-glass" reasoning in which a user can override access control based on urgency but thereby becomes subject to additional monitoring (e.g., CCTV) and post facto justification requirements. Marinovic et al. adopt multivalued reasoning to determine a suitable decision for the access control system.

Ingolfo et al. [2014] propose Nòmos 3, a goal-based framework to determine compliance of roles and requirements. They consider two norms (rights and duties) and propose two roles (social and legal). When a social role has to achieve a goal, the corresponding legal role must be compliant, i.e., it has to satisfy required preconditions and postconditions of norms. Their distinction of roles is not substantial. Ingolfo et al. provide a graphical modeling language for compliance, but it is limited and does not support defining dominance among norms. Ghanavati et al. [2014] provide heuristics for reading textual regulations to determine their meanings and potential dominance relations. They do not deal with instances and do not provide a formal approach.

Directions. Important directions include supporting dominance based on (1) organizational context, as in the legal principle of *Lex Superior*, which says that the decisions of a higher court take precedence over those of a lower court, and (2) temporal recency, as in the legal principle of *Lex Posterior*, which says that more recent decisions take precedence over earlier ones. Another direction is to provide methods in handling a cycle of dominance relations and in applying priority to a set of dominance relations.

Acknowledgments

We thank the US Department of Defense for support through the Science of Security Lablet grant to NC State University.

References

[Alberti *et al.*, 2006] Marco Alberti, Marco Gavanelli, Evelina Lamma, Federico Chesani, Paola Mello, and Paolo Torroni. Compliance verification of agent interaction: A logic-based software tool. *Applied Artificial Intelligence*, 20(2-4):133–157, February–April 2006.

[Alechina *et al.*, 2013] Natasha Alechina, Mehdi Dastani, and Brian Logan. Reasoning about normative update. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 20–26, Beijing, August 2013.

[Boella and van der Torre, 2008] Guido Boella and Leendert van der Torre. Institutions with a hierarchy of authorities in distributed dynamic environments. *Artificial Intelligence and Law*, 16(1):53–71, March 2008.

[Clarke *et al.*, 1986] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8:244–263, April 1986.

[Criado *et al.*, 2015] Natalia Criado, Elizabeth Black, and Michael Luck. A coherence maximisation process for solving normative inconsistencies. *Autonomous Agents and Multi-Agent Systems (JAAMAS)*, June 2015. Online.

[Dignum *et al.*, 2005] Frank Dignum, Jan Broersen, Virginia Dignum, and John-Jules Meyer. Meeting the deadline: Why, when and how. In Michael G. Hinchey, James L. Rash, Walter F. Truszkowski, and Christopher A. Rouff, editors, *Formal Approaches to Agent-Based Systems*, volume 3228 of *Lecture Notes in Computer Science*, pages 30–40. Springer, 2005.

[Gebser *et al.*, 2014] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Clingo* = ASP + control: Preliminary report. In Michael Leuschel and Tom Schrijvers, editors, *Technical Communications of the Thirtieth International Conference on Logic Programming (ICLP)*, volume 1433, 2014.

[Gelfond, 2008] Michael Gelfond. Answer sets. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, pages 285–316. Elsevier, 2008.

[Ghanavati *et al.*, 2014] Sepideh Ghanavati, Llio Humphreys, Guido Boella, Luigi Di Caro, Livio Robaldo, and Leendert van der Torre. Compliance with multiple regulations. In *Proceedings of the 33rd International Conference on Conceptual Modeling (ER)*, volume 8824 of *Lecture Notes in Computer Science*, pages 415–422. Springer, October 2014.

[Governatori *et al.*, 2007] Guido Governatori, Joris Hulstijn, Régis Riveret, and Antonino Rotolo. Characterising deadlines in temporal modal defeasible logic. In *AI 2007: Advances in Artificial Intelligence*, number 4830 in Lecture Notes in Computer Science, pages 486–496. Springer, 2007.

[Ingolfo *et al.*, 2014] Silvia Ingolfo, Ivan Jureta, Alberto Siena, Anna Perini, and Angelo Susi. Nòmos 3: Legal compliance of roles and requirements. In Eric Yu, Gillian Dobbie, Matthias Jarke, and Sandeep Purao, editors, *Conceptual Modeling*, volume 8824 of *Lecture Notes in Computer Science*, pages 275–288. Springer, 2014.

[Knobbout and Dastani, 2012] Max Knobbout and Mehdi Dastani. Reasoning under compliance assumptions in normative multiagent systems. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, pages 331–340, Valencia, 2012.

[Marinovic *et al.*, 2014] Srdjan Marinovic, Naranker Dulay, and Morris Sloman. Rumpole: An introspective break-glass access control language. *ACM Transactions on Information and System Security (TISSEC)*, 17(1):2:1–2:31, August 2014.

[Oren *et al.*, 2008] Nir Oren, Michael Luck, Simon Miles, and Timothy J Norman. An argumentation inspired heuristic for resolving normative conflict. In *Proceedings of the 5th Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems (COIN)*, pages 41–56, Toronto, 2008.

[Prakken and Sergot, 1996] Henry Prakken and Marek Sergot. Contrary-to-duty obligations. *Studia Logica*, 57(1):91–115, 1996.

[Singh, 2013] Munindar P. Singh. Norms as a basis for governing sociotechnical systems. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(1):21:1–21:23, December 2013.

[Vasconcelos *et al.*, 2009] Wamberto W. Vasconcelos, Martin J. Kollingbaum, and Timothy J. Norman. Normative conflict resolution in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 19(2):124–152, October 2009.

[Yolum and Singh, 2004] Pınar Yolum and Munindar P. Singh. Reasoning about commitments in the event calculus: An approach for specifying and executing protocols. *Annals of Mathematics and Artificial Intelligence*, 42(1):227–253, September 2004.