A Generalized Commitment Machine for 2CL Protocols and its Prolog Implementation

Matteo Baldoni, Cristina Baroglio, Federico Capuzzimati, Elisa Marengo, and Viviana Patti

Università degli Studi di Torino — Dipartimento di Informatica
c.so Svizzera 185, I-10149 Torino (Italy)

Abstract. In practical contexts where protocols model business interactions (e.g. trading, banking), designers need tools allowing them to analyse the impact on the possible interactions of regulations, preferences, conventions and the like. This work faces the issue of how to equip commitment protocols with formal and practical instruments aimed at supporting such an analysis by identifying the possible risks of violation and, thus, enabling the definition of operational strategies aimed at reducing risks of violation. Specifically, we present an operational semantics for the commitment protocol language 2CL as well as a tool for visualizing as a graph the possible interactions, labelling the states of the interaction so as to highlight legal situations and violations.

Keywords: Commitment-based Interaction Protocols, Constraints among Commitments, Commitment Machine, Risks of Violation

1 Introduction and Motivation

Agent interaction is generally specified by defining *interaction protocols* [18]. For communicating with one another, agents must follow the schema that the protocol shapes. Different protocol models can be found in the literature, this work concerns *commitment-based protocols* [16, 21]. This kind of protocols relies on the notion of commitment, which in turn encompasses the notions of debtor and creditor: when a commitment is not fulfilled, the debtor is liable for that violation but as long as agents reciprocally satisfy their commitments, any course of action is fine.

In many practical contexts where protocols model *business interactions* (e.g. trading, banking), designers must be able to regulate and constrain the possible interactions as specified by *conventions*, *regulations* [4], *preferences* or *habits*. Some proposals attack the issue of introducing similar regulations inside commitment protocols [3, 9, 6, 13], however, none of them brought yet to the realization of a *tool* that allows visualizing and analysing how regulations or constraints impact on the interactions allowed by a commitment-based protocol. The availability of intuitive and possibly graphical tools of this kind would support the identification of possible violations, thus enabling an *analysis of the risks* the interaction could encounter. As a consequence, it would be possible to raise alerts concerning possible violations before the protocol is enacted, and to reduce risks by defining proper operational strategies, like regimentation (aimed at preventing the occurrence of violations) or enforcement (introduction of warning mechanisms) [11].

The work presented in this paper aims at filling this gap. To this purpose, we started from the commitment protocol language 2CL described in [3], whose key characteristic is the extension of the regulative nature of commitments by featuring the definition of patterns of interaction as sets of *constraints*. Such constraints declaratively specify either conditions to be achieved or the order in which some of them should be achieved. The first contribution is, therefore, a formal, operational semantics for the proposal in [3], which relies on the Generalized Commitment Machine in [17]. We named our extension 2CL-Generalized Commitment Machines (2CL-GCM for short). On top of this, it was possible to realize the second contribution of this work: a Prolog implementation for 2CL-GCM, which extends the implementation in [19], and is equipped with a graphical tool to explore all the possible executions, showing both commitment and constraint violations. The implementation is part of a plug-in Eclipse which supports 2CL-protocol design and analysis.

The chief characteristic of our solution is that it performs a *state evaluation* of protocol constraints, rather than performing path evaluation (as, instead, done by model checking techniques). State evaluation allows considering each state only once, labelling it as a state of violation if some constraint is violated in it or as a legal state when no constraint is violated. This is a great difference with respect to path evaluation, where a state belonging to different paths can be classified as a state of violation or not depending on the path that is considered. The advantage is practical: state evaluation allows to easily supply the user an overall view of the possible alternatives of action, highlighting those which will bring to a violation and those that will not. State evaluation, however, is possible only by making some restriction on the proposal in [3]. Specifically, we assume that the domain is expressed in terms of positive facts only.

The paper is organized as follows. Section 2 briefly summarizes 2CL interaction protocol specification. Section 3 describes the formalization of 2CL-GCM. Section 4 presents a Prolog implementation of 2CL-GCM. Section 5 describes the 2CL Tools that supply features for supporting the protocol design and analysis. Related Work and Conclusions end the paper. Along the paper we use as a running example the well-known NetBill interaction protocol.

2 Background: 2CL Interaction Protocols

Let us briefly recall the chief characteristics of commitment protocols, as defined in [3]. In this approach, commitment protocols feature an explicit distinction between a *constitutive* and a *regulative* specification. The former defines the protocol actions, while the latter encodes the constraints the interaction should respect. Both specifications are based on *commitments*. Commitments are di-

	Relation	*Operator*	*Repr.*	*LTL formula*
Relation Operators	Correlation	*A correlate B*	$A \bullet\!\!- B$	$\Diamond A \supset \Diamond B$
		A not correlate B	$A \bullet\!\!\not- B$	$\Diamond A \supset \neg \Diamond B$
	Co-existence	*A co-exist B*	$A \bullet\!\!-\!\!\bullet B$	$A \bullet\!\!- B \wedge B \bullet\!\!- A$
		A not co-exist B	$A \bullet\!\!\not-\!\!\bullet B$	$A \bullet\!\!\not- B \wedge B \bullet\!\!\not- A$
Temporal Operators	Response	*A response B*	$A \bullet\!\!-\!\!\triangleright B$	$\Box(A \supset \Diamond B)$
		A not response B	$A \bullet\!\!\not-\!\!\triangleright B$	$\Box(A \supset \neg \Diamond B)$
	Before	*A before B*	$A -\!\!\triangleright\!\!\bullet B$	$\neg B \cup A$
		A not before B	$A \not-\!\!\triangleright\!\!\bullet B$	$\Box(\Diamond B \supset \neg A)$
	Cause	*A cause B*	$A \bullet\!\!-\!\!\triangleright\!\!\bullet B$	$A \bullet\!\!-\!\!\triangleright B \wedge A -\!\!\triangleright\!\!\bullet B$
		A not cause B	$A \bullet\!\!\not-\!\!\triangleright\!\!\bullet B$	$A \bullet\!\!\not-\!\!\triangleright B \wedge A \not-\!\!\triangleright\!\!\bullet B$

Table 1. 2CL operators and their meaning.

rected from a debtor to a creditor. The notation $C(x, y, r, p)$ denotes that agent x commits to an agent y to bring about the consequent condition p when the antecedent condition r holds. When r equals *true*, we use the short notation $C(x, y, p)$. The interacting partners share a *social state* that contains commitments and other facts that are relevant to their interaction. Every partner can affect the social state by executing actions, whose definition is given in terms of operations onto the social state, see [21]. The partners' behaviour is affected by commitments, which have a *regulative* nature, in that debtors should act in accordance with the commitments they have taken.

Definition 1 (Interaction protocol). *An interaction protocol $\mathcal{P}$ is a tuple $\langle Ro, F, s_0, A, C\rangle$, where Ro is a set of roles, identifying the interacting parties, F is a set of facts and commitments that can occur in the social state, s_0 is the set of facts and commitments in the initial state of the interaction, A is a set of actions, and C is a set of constraints.*

The set of social actions A, defined on F and on Ro, forms the *constitutive specification* of the protocol. The social effects are introduced by the construct *means*, and their achievement can depend on a precondition (conditional effects). Both preconditions and effects are given in terms of the set F specified in the protocol, which contains commitments and facts (i.e. the conditions that are brought about). The means construct amounts to a *counts-as* relation [15, 11]. For instance, consider the action *sendGoods* reported in Table 2. Its social meaning is that it makes the facts *goods* true (the goods were delivered to the customer) and creates the commitment $\mathsf{C}(m, c, pay, receipt)$ that corresponds to a promise by the merchant to send a receipt after the customer has paid. Further examples can be found in the first part of Table 2, which reports all the actions of the NetBill protocol. The formalization is inspired by those in [21, 19].

The *regulative specification* of the protocol is made of the set of 2CL constraints C, defined on F and on Ro as well. 2CL is a declarative language, which allows expressing what is mandatory and what is forbidden without the need of listing the possible executions extensionally. Constraints have the form "dnf_1 op dnf_2", where dnf_1 and dnf_2 are disjunctive normal forms of facts and

Action Definitions

(a1) $\mathit{sendRequest}$ **means** $\mathit{request}$ **if** $\neg \mathit{quote} \wedge \neg \mathit{goods}$

(a2) $\mathit{sendQuote}$ **means** $\mathit{quote} \wedge \mathit{create}(\mathsf{C}(m, c, \mathsf{C}(c, m, \mathit{goods}, \mathit{pay}), \mathit{goods}))$
$\wedge\ \mathit{create}(\mathsf{C}(m, c, \mathit{pay}, \mathit{receipt}))$

(a3) $\mathit{sendAccept}$ **means** $\mathit{create}(\mathsf{C}(c, m, \mathit{goods}, \mathit{pay}))$ **if** $\neg \mathit{pay}$

(a4) $\mathit{sendGoods}$ **means** $\mathit{goods} \wedge \mathit{create}(\mathsf{C}(m, c, \mathit{pay}, \mathit{receipt}))$

(a5) $\mathit{sendEPO}$ **means** pay

(a6) $\mathit{sendReceipt}$ **means** $\mathit{receipt}$ **if** pay

Constraints

(c1) $\mathit{quote} \multimap\!\!\bullet\ \mathsf{C}(c, m, \mathit{goods}, \mathit{pay}) \vee \mathsf{C}(c, m, \mathit{pay})$

(c2) $\mathsf{C}(m, c, \mathit{pay}, \mathit{receipt}) \wedge \mathit{goods} \multimap\!\!\bullet\ \mathit{pay}$

(c2) $\mathit{pay} \bullet\!\!\multimap\!\!\bullet\ \mathit{receipt}$

Table 2. Actions and constraints for the NetBill protocol: m stands for merchant while c stands for customer.

commitments, and op is one of the 2CL operators, reported in Table 1 together with their Linear-time Temporal Logic [8] interpretation and with their graphical notation. Basically, there are two kinds of operators: *relational* and *temporal*. The former kind expresses constraints on the co-occurrence of conditions (if this condition is achieved then also that condition must be achieved, but the order of the two achievements does not matter). For instance, one may wish to express that both the payment for some item and its delivery must occur without constraining the order of the two conditions: no matter which occurs first, when one is met, also the other must be achieved. Temporal operators, instead, capture the relative order at which different conditions should be achieved. The second part of Table 2 reports the constraints imposed by the NetBill protocol: *(c1)* means that a quotation for a price must occur before a commitment to pay or a conditional commitment to pay given that some goods were delivered; *(c2)* that the conditional commitment to send a receipt after payment and the delivery of goods must occur before the payment is done; *(c3)* that after payment a receipt must be issued and if a receipt is issued a payment must have occurred before.

Among the *possible* interactions, derivable from the action specification, those that respect the constraints are said to be *legal*. Violations amounting to the fact that a constraint is not respected can be detected *during the execution*. The following section provides the operational semantics 2CL lacked of.

3 2CL Generalized Commitment Machine

In order to provide the semantics of commitment protocols as specified in [3] (see Definition 1 of this paper), we define the 2CL *generalized commitment machine* (2CL-GCM). Briefly, 2CL-GCM relies on the notion of *generalized commitment machine* (GCM) (introduced in [17]) for what concerns the inference of the possible evolutions of the social state, that can be obtained by taking into account only the protocol actions and the commitment life cycle. Additionally, 2CL-GCM also accounts for 2CL constraints.

According to [17], a GCM features a set S of possible *states*, each of which is represented by a logical expression of facts and commitments. S represents the possible configurations of the social state.

Example 1. Considering NetBill, the expression $goods \wedge \mathsf{C}(c, m, pay)$ represents one possible configuration of the social state. This expression means that goods were shipped and that there is a commitment from c (customer) to m (merchant) to pay for them.

Particularly relevant is the subset of S, whose elements are named *good states*: they are the desired final states of the interaction. The characterization of good states depends on the particular application. For instance, they may be only those that do not contain unsatisfied active commitments, or they could be the ones which satisfy a condition of interest (e.g. payment done and goods shipped).

In GCM, transitions between the states are logically inferred on the basis of an action theory Δ, that contains a set of axioms of the kind $p \overset{a}{\hookrightarrow} q$, meaning that q is a consequence of performing action a in a state where p holds. When q is false the meaning is that a is impossible if p holds. In general, Δ contains all the axioms deriving from the specification of the protocol actions. Additionally, Δ also contains an axiom for each action a and for each couple of states s and s' such that the execution of a in s causes a transition to s': for instance, if the precondition p of a is satisfied in s and its effect q is satisfied in s', then $s \overset{a}{\hookrightarrow} s'$ is in Δ. The way in which these axioms are obtained is explained in [17].

Example 2. According to the 2CL protocol syntax, the action *sendAccept*, performed by the customer to accept a quote of the merchant, is defined as *sendAccept* **means** CREATE(C(c,m,goods,pay)) **if** ¬ pay. The corresponding axiom is $\neg pay \overset{sendAccept}{\hookrightarrow} \mathsf{C}(c, m, goods, pay)$. Now, if one considers a state in which $\neg pay \wedge quote$ holds, it is also possible to infer the axiom $\neg pay \wedge quote \overset{sendAccept}{\hookrightarrow} \mathsf{C}(c, m, goods, pay)$.

In GCM paths must be infinite. All the finite paths are transformed into infinite ones by adding a transition from the last state of the finite path towards an artificial new state with a self loop [17]. In 2CL-GCM we adopt the same assumption and the same mechanism for transforming finite paths into infinite ones. We are now ready to define 2CL-GCM. The definition adopts the same notation in [17].

Definition 2 (2CL Generalized Commitment Machine). *Let $\vdash$ and $\equiv$ be, respectively, the logical consequence and the logical equivalence of propositional logic. A 2CL-GCM is a tuple $\mathbf{P} = \langle S, A, s_0, \Delta, G, C \rangle$, where:*

- *S is a finite set of states;*
- *A is a finite set of actions;*
- *$s_0 \in S$ is the initial state;*
- *Δ is an action theory;*

- *$G \subseteq S$ is a set of good states;*
- *C is a set of* 2CL *constraints.*

(i) *Members of S are logically distinct, that is: $\forall s, s' \in S, s \not\equiv s'$;* (ii) false $\notin S$*; and* (iii) $\forall s \in G, s' \in S : (s' \vdash s) \Rightarrow (s' \in G)$*, i.e. any state that logically derives a good state is also good.*

A sequence of states is a *path* of a 2CL-GCM if it satisfies all of the constraints in C. Since 2CL constraints are defined in terms of LTL formulas, to perform the verification one can consider the *transition system* corresponding to the path. Given a sequence of states interleaved by actions, the corresponding transition system can be derived quite straightforwardly. Intuitively, the set of states and transitions of the system is the same set of states and transitions in the sequence. A requirement on transition systems is that each state has at least one outgoing transition (i.e. runs are infinite).

Definition 3 (Transition System). *Let $\tau = \langle(\tau_0, a_0, \tau_1), (\tau_1, a_1, \tau_2), \ldots\rangle$ be an infinite and ordered sequence of triples, where τ_i is the state at position i in τ and a_i is the action that causes the transition from state τ_i to state τ_{i+1}. The transition system $T(\tau)$ corresponding to τ is a triple $\langle S_\tau, \delta_\tau, L_\tau\rangle$ where:*

- *$S_\tau = \{\tau_i |\ \tau_i \in \tau\}$ is a set of states;*
- *$\delta_\tau : S_\tau \to S_\tau$ is a transition function where: $\delta(\tau_j) = \tau_k$ iff $(\tau_j, a, \tau_k) \in \tau$;*
- *$L : S_\tau \to 2^F$ is a labelling function, where: F is a set of facts and commitments and given $l \in F$, then $l \in L(\tau_i)$ iff $\tau_i \vdash l$.*

To define a 2CL-GCM path, we adapt the definition of GCM path by adding the requirement that the sequence of states satisfies all the constraints of the 2CL-GCM. This condition is checked on the transition system corresponding to the path, by means of the LTL satisfaction relation [1]. We denote it with the symbol $\models_{LTL}$. In the following definition we adopt the same notation in [17].

Definition 4 (2CL-GCM path). *Let $\mathbf{P} = \langle S, A, s_0, \Delta, G, C\rangle$ be a* 2CL-GCM*. Let $\tau = \langle(\tau_0, a_0, \tau_1), \ldots\rangle$ be an infinite sequence of triples and $T(\tau)$ be the corresponding transition system. Let $inf(\tau)$ be the set of states that occur infinitely often in τ. τ is a path generated from $\mathbf{P}$ when:*

(i) $\forall(\tau_i, a_i, \tau_{i+1})$ in τ then $\tau_i, \tau_{i+1} \in S$ and $a_i \in A$ and $\tau_i \overset{a_i}{\hookrightarrow} \tau_{i+1} \in \Delta$; and
(ii) $inf(\tau) \cap G \neq \emptyset$; and
(iii) $\forall c \in C :\ T(\tau), \tau_0 \models_{LTL} c$

In the above definition, *(i)* and *(ii)* are the conditions for a path to be generated from a GCM [17]. Condition *(i)* requires that each *state* in the sequence is a state of the 2CL-GCM, that the *action* that causes the transition from a state to the subsequent one in the sequence is an action of the 2CL-GCM, and that the transition is inferable according to the *axioms* in Δ. It also requires that the path is *infinite*. Condition *(ii)* requires that at least one *good state* occurs infinitely often in the sequence. Condition *(iii)* was added to account for the evaluation of the protocol constraints. According to the LTL semantics, the notation $\mathcal{M}, s \models_{LTL} \varphi$ means that every execution path π of $\mathcal{M}$, starting at s, is such that $\pi \models_{LTL} \varphi$. Since $T(\tau)$ is a transition system made only of *one* linear path (by construction), whose starting state is the starting state of τ, the condition $T(\tau), \tau_0 \models_{LTL} c$ amounts to checking if c is satisfied in the path of the transition system, corresponding to τ.

Given a protocol specification it is possible to build the corresponding 2CL-GCM:

Definition 5 **(2CL-GCM of a protocol).** *Let $\mathcal{P} = \langle Ro, F, s_0, A, C\rangle$ be a protocol, S be a set of states and $G \subseteq S$ be a set of good states. $\mathbf{P} = \langle S, L_A, s_0, \Delta, G, C\rangle$ is a 2CL-GCM of $\mathcal{P}$ when (i) L_A is the set of action labels in A; and (ii) Δ is the action theory of A, i.e.:*

- *for each (a **means** e **if** p) belonging to A, then $p \overset{a}{\hookrightarrow} e$ belongs to Δ;*
- *Δ is closed under inference rules in [17].*

Since the state s_0 and the constraints C of a 2CL-GCM are the same of the protocol, the definition uses the same symbols. By varying the sets S and G different 2CL-GCMs associated to the same protocol are obtained: when S contains all the states that can be reached from s_0, applying the protocol actions, the machine can infer all the possible interactions; when S is smaller, only a subset of the possible interactions is determined.

4 Implementation of the 2CL Commitment Machine

This section describes a Prolog implementation for the 2CL-GCM, formalized above. It allows exploring all the possible executions of an interaction protocol, showing the *regulative violations*— i.e. both those states in which some constraint is violated and those that contain unsatisfied commitments. We prove that if a path is legal according to the implementation, then it is a path of the corresponding 2CL-GCM.

The implementation is realized in *tuProlog*[1] and it builds upon the *enhanced commitment machine* realized by Winikoff et al. [19]. By relying on it, we inherit the mechanisms for the computation of the possible interactions. Specifically, enhanced commitment machines feature the generation of the reachable states, the transitions among them and the management of commitments (like the operations of discharge, creation and so on). Our extension equips them with the possibility of evaluating 2CL constraints. The aim is to provide a qualitative view of the possible interactions, highlighting those that violate some constraints. The interacting parties are not prevented from entering in illegal paths (autonomy is preserved), but they are made aware of the risks they are encountering and that they may incur in penalties as a consequence of the violations they caused [4].

In order to provide a compact but global view of the possible interactions, the evaluation of constraints is performed on *one state at a time* rather than on paths (as, instead, usually done in LTL model checking). Specifically, the

[1] `http://www.alice.unibo.it/xwiki/bin/view/Tuprolog/`

Relation	*State Condition*
Correlation	$\psi(A \bullet\!\!- B) = A \wedge B$
	$\psi(A \bullet\!\!\not- B) = \neg(A \wedge B)$
Co-existence	$\psi(A \bullet\!\!-\!\!\bullet B) = \psi(A \bullet\!\!- B) \ \wedge \psi(B \bullet\!\!- A)$
	$\psi(A \bullet\!\!\not-\!\!\bullet B) = \psi(A \bullet\!\!\not- B) \ \wedge \psi(B \bullet\!\!\not- A)$
Response	$\psi(A \bullet\!\!-\!\!\triangleright B) = A \wedge B$
	$\psi(A \bullet\!\!\not-\!\!\triangleright B) = \neg(A \wedge B)$
Before	$\psi(A -\!\!\triangleright\!\!\bullet B) = \neg(B \wedge \neg A)$
	$\psi(A \not-\!\!\triangleright\!\!\bullet B) = \neg(A \wedge B)$
Cause	$\psi(A \bullet\!\!-\!\!\triangleright\!\!\bullet B) = \psi(A \bullet\!\!-\!\!\triangleright B) \ \wedge \psi(A -\!\!\triangleright\!\!\bullet B)$
	$\psi(A \bullet\!\!\not-\!\!\triangleright\!\!\bullet B) = \psi(A \bullet\!\!\not-\!\!\triangleright B) \ \wedge \psi(A \not-\!\!\triangleright\!\!\bullet B)$

Table 3. State conditions corresponding to 2CL operators.

state content is given in terms of positive facts and commitments. A fact that is not true in a state has not been achieved yet, so we use negation as failure in the conditions of the action definitions to verify whether a fact is present or not in the social state. In this setting, the evaluation of 2CL constraints can be made on single states. For instance, if in a state b holds but a does not, we can infer that the constraint '*a before b*' is violated. This kind of verification, however, can be performed only on a subset of 2CL formulas, specifically, only on constraints corresponding to conditions that persist (i.e. that involve DNFs of facts without negation). Since commitments do not persist because they can be cancelled, discharged, etc., another requirement is to *associate a fact to each operation* that is performed on commitments (along the line of [12]). These facts are automatically asserted whenever an operation is performed on a commitment and they can be used in constraint formulas. For instance, when a commitment $C(x, y, r, p)$ is created, the fact CREATED$(C(x, y, r, p))$ is added to the state, and so forth for the other operations. Notice that these facts are not meant to associate to each commitment its current state in the commitment life cycle, as instead done in [5]. This information can be inferred from the presence or absence of the commitment in the state.

Given a constraint c, we denote by $\psi(c)$ the corresponding condition to be verified one state at a time (*state condition*). The above assumptions allow the simplification of the LTL formulas, corresponding to the 2CL operators, in the way that is reported in Table 3. Consider, for instance, the *before* operator ($-\!\!\triangleright\!\!\bullet$): it requires that A is met before or in the same state of B. So, given a run π, if in π there is a state j such that B holds while A does not, that is a state where a violation occurred. In formulas: $\pi_i \models_{LTL} A -\!\!\triangleright\!\!\bullet B \ \Leftrightarrow \ \neg\exists j \geq i \ s.t. \ \pi_j \models_{LTL} (B \wedge \neg A)$ (when a formula does not contain temporal operators, the relation $\models_{LTL}$ checks the condition in the first state of a path).

The other 2CL operators can be divided in two groups. *Correlation* ($\bullet\!\!-$) and *response* ($\bullet\!\!-\!\!\triangleright$) are part of the same group. $A \bullet\!\!- B$ requires that if A is achieved in a run, then also B is achieved in the same run (before or after A is not relevant). If B is achieved before A it will remain true also after. Therefore, in those cases in which the constraint is satisfied, from a certain time onwards both conditions will hold. In formulas: $\pi_i \models_{LTL} A \bullet\!\!- B \Leftrightarrow \neg\exists j \geq i\ s.t.\ \pi_j \models_{LTL} A$ and $\forall j' \geq j,\ \pi_{j'} \models_{LTL} (A \wedge \neg B)$. The same equivalence holds for $\pi_i \models_{LTL} A \bullet\!\!-\!\!\triangleright B$. In 2CL $A \bullet\!\!-\!\!\triangleright B$ requires that when A is met, B is achieved at least once later (even if it already occurred in the past) but under our assumptions it can be checked in the same way of correlation. The state condition amounts to verifying whether a state satisfies A but does not satisfy B. Notice that states that satisfy the test cannot be marked as states of violation because the constraint does not require B to hold *whenever* A holds. A state of violation is signalled when the interaction does not continue after it: we say that there is a *pending* condition.

Negated correlation, *response* and *before* correspond to the same formula: $\pi_i \models_{LTL} A\ \mathsf{op}\ B \Leftrightarrow \neg\exists j \geq i\ s.t.\ \pi_j \models_{LTL} (A \wedge B)$ where $\mathsf{op} \in \{\bullet\!\!\not-, \bullet\!\!\not-\!\!\triangleright, \not-\!\!\triangleright\!\!\bullet\}$. Intuitively, a constraint of the kind $A \bullet\!\!\not- B$ (negative correlation) requires that if A holds, B is not achieved. Since facts persist, this amounts to check that the two conditions do not hold in the same state, otherwise a violation occurs. *Negative response* (*negative before*) adds a *temporal aspect* to not-correlation: if A holds, B cannot hold later (before, respectively). Since facts persist, the first achieved condition will remain true also after the other becomes true. Also in this case we only need to check that the two conditions do not hold together.

Derived operators are decomposed and the reasoning made for the operators, from which they derive, is applied. For instance, *cause* ($\bullet\!\!-\!\!\triangleright\!\!\bullet$) derives from *before* and *response*. If a state does not satisfy the response part of the cause, it is marked as "pending"; if it violates the before part, it is marked as a "violation". Both labels are applied when the state does not satisfy any of the two.

Summarizing, given a constraint formula and a state in which to verify it, we have three possible cases: (i) the state satisfies the formula; (ii) the state does not satisfy the formula and this leads to a violation; and (iii) the state does not satisfy the formula but the violation is potential, depending on future evolution. Considering all the constraints of a protocol, a state can both violate some constraint and have pending conditions. Moreover, states are also evaluated based on the presence of unsatisfied active commitments.

These considerations enable the generation and the labelling of all the states that can be reached by applying the protocol actions. The result is a labelled graph, as defined in Definition 6, where each state is associated a set of labels.

Definition 6 (Labelled Graph). *Let $\mathcal{P} = \langle Ro, F, s_0, A, C\rangle$ be a protocol, the corresponding labelled graph $G(\mathcal{P})$ is a triple (S, δ, L) where:*

- *S is a set of states reachable from s_0, such that $\forall s, s' \in S, s \not\equiv s'$;*
- *$\delta \subseteq S \times A \times S$ is a transition relation such that $\forall (s, a, s') \in \delta$ then $s, s' \in S$ and $\exists a \in A$ s.t. when a is executed in s it determines s';*
- *$L \subseteq S \times \{$pending,violation,final,non-final$\}$ is a labelling relation such that given $s \in S$:*
  - *violation $\in L(s)$ iff $\exists c \in C$ s.t. $s \not\models_{LTL} \psi(c)$ and c is not a response or a correlation;*
  - *pending $\in L(s)$ iff $\exists c \in C$ s.t. $s \not\models_{LTL} \psi(c)$ and c is a response or a correlation;*
  - *final $\in L(s)$ iff there are no unsatisfied active commitments in s;*

```
explore(StateNum,Free,NextFree) :-
  state(StateNum,State,_),
  findall(t(StateNum,A,S2),nextstate(State,A,S2),Ts),
  add_states(Ts,Free,NextFree), add_transitions(Ts).

add_states([],N,N).
add_states([t(_,_,S)|Ss],N,N1) :-
  state(_,St,_), seteq(St,S), !, add_states(Ss,N,N1).
add_states([t(_,_,S)|Ss],N,N3) :-
  labels(S,L), assert(state(N,S,L)),
  N1 is N+1, explore(N,N1,N2), add_states(Ss,N2,N3).

labels(State,Labels) :- find_labels(State,[],Labels).

find_labels(S,L1,R) :-
  check_violation(S,L1,L2),
  check_pending(S,L2,L3),
  check_commitments(S,L3,R).

check_pending(State,L,[pending(Constr)|L]) :-
  response(A,B,Constr),
  consequence(A,State), \+consequence(B,State).

check_violation(State,L,[violation(Constr)|L]) :-
  before(A,B,Constr),
  consequence(B,State), \+consequence(A,State).

check_commitments(State,L,[final|L]) :-
  \+member(c(_,_,_),State).
check_commitments(State,L,[non-final|L]) :-
  member(c(_,_,_),State).
```

Listing 1.1. Prolog clauses that generate the labelled graph: consequence(A,State) is a clause that determines if the fact (or the DNF formula) A can be derived in State; response and before are constraints. The complete program is downloadable at the URL http://di.unito.it/2cl.

- *not-final* $\in L(s)$ *iff* s *contains unsatisfied active commitments.*

Following Definition 6, our implementation starts from the initial state and determines all the reachable states, by applying a depth-first search, as in [19]. The difference is that our representation of the states contains also a list of labels, which identify the presence of active commitments and of pending or violated conditions. Listing 1.1 reports part of the Prolog program that generates the labelled graph. The mechanism is as follows: given a state, *explore* finds the set of the possible successors by applying the effects of the actions, whose preconditions are satisfied in the state. A state is added only if it is new (not explored yet). Before adding it, *find_labels* considers all the constraints and checks them on the state. Constraints are represented as *constraint*(A, B, Id), where *constraint* is the 2CL operator used by the constraint, A and B are the antecedent and the consequent conditions of the constraint, and Id is the identifier of the constraint. Listing 1.1 reports, as an example of tests performed on states, the verification of a *response* and of a *before*. The clause *check_pending*, that is reported here, verifies response constraints: it is satisfied if there is a constraint of kind response, whose antecedent condition can be derived in the state, while the consequent condition cannot. In this case, the label *pending* is added to the list of labels of the state. A similar clause checks the correlation constraint. Instead, the clause *check_violation*, checks before constraints, which are violated if the consequent condition can be derived in the state while their antecedent cannot. Other similar clauses, checking different conditions, are defined for the other operators. Finally, the program checks the presence of unsatisfied commitments (*check_commitments*) and adds the label *final* or *not-final* consequently. The result of running this program on a protocol specification is a graph of the reachable annotated states. Annotations follow the graphical convention explained in Section 5.

Given a labelled graph we are now able to define when a path is legal.

Definition 7 (Legal path). *Let $G(\mathcal{P}) = (S, \delta, L)$ be a labelled graph, $\pi = \langle(\pi_0, a_0, \pi_1), \ldots, (\pi_{n-1}, a_{n-1}, \pi_n)\rangle$ be a path of at least one state. π is a legal path of $G(\mathcal{P})$ when:*

(i) $\forall i \geq 0, \pi_i$ is a state of the graph and $(\pi_i, a_i, \pi_{i+1}) \in \delta$.
(ii) $\nexists i \geq 0$ such that $\pi_i \in \pi$ and violation $\in L(\pi_i)$;
(iii) pending $\notin L(\pi_n)$ and $final \in L(\pi_n)$.

Condition (i) requires that the transitions in the path find correspondence in the graph; (ii) requires that none of the states of the path violates a constraint; (iii) requires that the last state of the path does not contain pending conditions or unsatisfied commitments.

Given a legal path π, of a labelled graph produced by Listing 1.1, we can prove its correctness w.r.t. the 2CL-GCM built on the same protocol specification. This actually corresponds to prove that π is a path of the 2CL-GCM.

Theorem 1 (Soundness). *Let $\mathcal{P} = \langle Ro, F, s_0, A, C\rangle$ be a protocol; let $G(\mathcal{P})$ be the corresponding labelled graph; let $\pi = \langle(\pi_0, a_0, \pi_1), \ldots, (\pi_{n-1}, a_{n-1}, \pi_n)\rangle$ be a path; and let $\mathbf{P} = \langle S_\pi, A, s_0, \Delta, G, C\rangle$ be the* 2CL-GCM *of $\mathcal{P}$, where S_π is the set of states in π and G is the set of states in π that do not contains unsatisfied commitments. If π is a legal path of G then π is a path of $\mathbf{P}$.*

The proof is by contradiction. It is omitted for lack of space.

5 2CL Tool for Protocol Design and Analysis

Let us now present the tool that we developed based on the technical framework, described in the previous sections. The tool supports the user in two different ways: (i) it features two graphical editors for specifying the protocol actions and the constraints; (ii) it generates different kinds of graphs for supporting the user in the analysis of the possible interactions and in understanding which of them are legal. The system is realized as an Eclipse plug-in, available at the URL `http://di.unito.it/2cl`.

The functionalities that the system supports can be grouped into three components: *design*, *reasoning* and *visualization*.

Design Component. The design component provides the tools that are necessary for defining the protocol. It supplies two editors: one for the definition of the actions and one for the definition of constraints. The *action definition* editor is basically a text editor, where actions can be specified following the grammar in [4]. The *regulative specification editor* allows the user to graphically define a set of constraints. Constraints are represented by drawing facts, connecting them with 2CL arrows (following the graphical representation of Table 1) or with logical connectives so as to design DNF formulas. The advantage of having a graphical editor is that it supplies a global view of constraints, thus giving the perception of the *flow* imposed by them, without actually specifying any rigid sequence (no-flow-in-flow principle [2]).
Reasoning Component. The reasoning component consists of a Java Parser and of the Prolog implementation of the commitment machine described in Section 4. The former generates different kinds of graphs as well as the Prolog program corresponding to the protocol specification. The latter is the input of the Prolog implementation of the commitment machine for the generation of the labelled graph. As explained, the labelled graph represents all the possible interactions where each state is labelled according to the evaluation of the protocol constraints. The graphical conventions is: (i) a state of violation is represented as a red diamond, with an incoming red arrow (e.g. states 54, 57, 108 in Figure 1); (ii) a state in which there is a pending condition is yellow (e.g. states 45, 53, 108); (iii) a state with a single outline, independently from the shape (e.g. 49, 57, 60), is a state that contains unsatisfied commitments; (iv) a state with a double outline, independently from the shape, does not contain active commitments (e.g. 41, 108). Graphical notations can be combined, e.g. a yellow diamond with single outline is a state where there are unsatisfied active commitments, where a constraint is violated and where there is a pending condition (e.g. 53, 57, 114).
Visualization Component. All the graphs produced by the reasoning component can be visualized as images. *Labelled graph*, however, can be explored by means of the tool *Graph Explorer*, which is realized in Java and relies on iDot (Incremental Dot Viewer) – an open source project that uses the prefuse[2] visualization framework for Dot graph display. The Graph Explorer supplies different functionalities, like the visualization of the shortest path given a source and a target state, and the visualisation of legal (or illegal) paths only. The user can add or delete a node in a path; search a state starting from its label; and search all the states that contain a certain fact or commitment. Moreover, the tool allows the exploration of the graph one state at a time, by choosing which node to expand. Figure 1 reports part of the labelled graph for NetBill.

Protocol Analysis. The tool can be used as a support in protocol analysis [4]. Particularly interesting is the possibility of exploring the labelled graph by means of the Graph Explorer, which can be used to predict whether performing a certain sequence of actions results in a violation and, in this case, if there is a way to return on a legal path. For what concerns the designer, it is not always easy,

[2] `http://prefuse.org/`

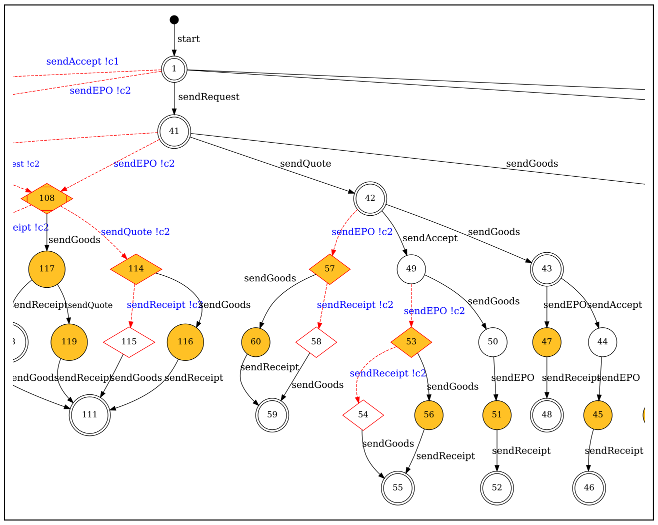

Fig. 1. Part of NetBill labelled Graph.

when specifying a protocol, to individuate which constraints to introduce but, with the help of the tool, it becomes easy to identify misbehaviours and revise the constraints so as to avoid them. Moreover, a designer can decide, by analysing the graph, to modify the specification so as to regiment some of the patterns expressed as constraints, or to remove some of them. For instance, considering the running example, from Figure 1 it is possible to infer that the protocol does not allow the customer to pay (*sendEPO*) before the merchant sends the goods. This is due to the constraint $\text{CREATED}(\mathsf{C}(m, c, pay, receipt)) \wedge goods \,{-}\!\!\triangleright\!\!\bullet\, pay$. If this behaviour was not in the intention of the designer, he/she can discover it and, e.g., relax the *before* constraint ($-\!\!\triangleright\!\!\bullet$) transforming it into a *co-existence* ($\bullet\!\!-\!\!\bullet$). If, instead, that is exactly the desired behaviour, one may decide to regiment *sendEPO* so as to enable the payment only after the goods have been sent.

The complete NetBill protocol encoding and the corresponding labelled graph together with further examples (like CNet, OECD guidelines and MiFID [4]) are available at `http://di.unito.it/2cl` (section Examples).

6 Related Work and Conclusions

This work provides an operational semantics of 2CL protocols [3], based on an extension of the Generalized Commitment Machine [17], and describes a Prolog implementation of this formalization, where the constraint evaluation is performed thanks to state conditions rather than by considering paths. The provided formalization allows the creation of compact and annotated graphs, which provide a global overview of the possible interactions, showing which are legal and which cause constraint (or commitment) violations. The aim was to support a tool, which enables the verification of exposure to risk on the graph of the possible executions, and taking decisions concerning how to behave or to modify the protocol in order to avoid such a risk. Due to this aim, we decided to base our implementation on [19], rather than on formalizations which support, for instance, model checking. The reason is that this work already is along the same line of ours, the intent being to give a global view on desirable and undesirable states. Winikoff et al. [19], however, propose to cope with undesired paths or undesired final states by adding ad-hoc preconditions to the actions, or by adding active commitments to states that are desired not to be final. This, however, complicates the reuse and the adaptation of the specification to different domains. On the contrary, the proposal in [3] results to be easily adaptable and customizable so as to address different needs of different domains, and it also allows for the specification of more expressive patterns of interaction, given as 2CL constraints.

Other approaches, are based on expectations and abductive logic programming. Chesani et al. [5] focus on the verification of *a-posteriori* trace compliance with respect to the specification. Montali et al. [14], instead, face the verification of static properties on the specification, stopping the verification when a counterexample that falsifies the property is found. None of these works, however, provide a global overview of the possible interactions and of potential risks of violation.

Concerning model checking, El-Menshawy et al. [7] propose a branching-time logic that extends CTL*, used to give a logical semantics to the operations on commitments. This approach was designed to perform verifications on commitment-protocol ruled interactions by exploiting symbolic model checking techniques. The properties that can be verified are those that are commonly checked in distributed systems: fairness, safety, liveness, and reachability. It would be interesting to integrate in this logical framework the 2CL constraints in order to combine the benefits of both approaches: on the one hand, the possibility to embed in the protocols expressive regulative specification, and, on the other hand, the possibility to exploit the logical framework to perform the listed verifications.

For what concerns the semantics of commitment protocols, the literature proposes different formalizations. Some approaches present an operational semantics that relies on commitment machines to specify and execute protocols [21, 20, 19]. Some others, like [9], use interaction diagrams, operationally specifying commitments as an abstract data type, and analysing the commitment's life cycle as a trajectory in a suitable space. Further approaches rely on temporal logics to give a formal semantics to commitments and to the protocols defined upon them. Among these, Giordano et al. [10] uses DLTL. All these approaches allow the inference of the possible executions of the protocol, but, differently than [3], all of them consider as the only regulative aspect of the protocol the regulative value of the commitments.

References

1. C. Baier and J.-P. Katoen. *Principles of Model Checking.* MIT Press, 2008.

2. M. Baldoni, C. Baroglio, and E. Marengo. Behavior-Oriented Commitment-based Protocols. In *Proc. of ECAI*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pp. 137–142. IOS Press, 2010.
3. M. Baldoni, C. Baroglio, E. Marengo, and V . Patti. Constitutive and Regulative Specifications of Commitment Protocols: a Decoupled Approach. *ACM Trans. on Int. Sys. and Tech., Spec. Iss. on Agent Communication*, 2011.
4. M. Baldoni, C. Baroglio, E. Marengo, and V. Patti. Grafting Regulations into Business Protocols: Supporting the Analysis of Risks of Violation. In A. Antón, D. Baumer, T. Breaux, and D. Karagiannis, editors, *RELAW 2011*, pp. 50–59, Trento, Italy, August 30th 2011. IEEE Xplore.
5. F. Chesani, P. Mello, M. Montali, and P. Torroni. Commitment Tracking via the Reactive Event Calculus. In C. Boutilier, editor, *IJCAI*, pp. 91–96, 2009.
6. A. K. Chopra and M. P. Singh. Constitutive Interoperability. In L. Padgham, D. C. Parkes, J. Müller, and S. Parsons, editors, *Proc. of AAMAS 2008*, volume 2, pp. 797–804, Estoril, Portugal, May 2008. IFAAMAS.
7. M. El-Menshawy, J. Bentahar, and R. Dssouli. Verifiable Semantic Model for Agent Interactions Using Social Commitments. In M. Dastani, A. Seghrouchni El Fallah, J. Leite, and P. Torroni, editors, *LADS 2009*, LNCS 6039, pp. 128–152, Torino, Italy, September 2010. Springer.
8. E. A. Emerson. *Temporal and Modal Logic*, volume B. Elsevier 1990.
9. N. Fornara and M. Colombetti. A Commitment-Based Approach To Agent Communication. *Applied Artificial Intelligence*, 18(9-10):853–866, 2004.
10. L. Giordano, A. Martelli, and C. Schwind. Specifying and Verifying Interaction Protocols in a Temporal Action Logic. *J. of Applied Logic*, 5(2):214–234, 2007.
11. A. J. I. Jones and M. Sergot. *On the Characterization of Law and Computer Systems: the Normative Systems Perspective*, pp. 275–307. John Wiley&Sons, 1994.
12. Ashok U. Mallya and Munindar P. Singh. Modeling Exceptions via Commitment Protocols. In F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh, and M. Wooldridge, editors, *AAMAS 2005*, pp. 122–129. ACM, 2005.
13. E. Marengo, M. Baldoni, C. Baroglio, A. K. Chopra, V. Patti, and M. P. Singh. Commitments with Regulations: Reasoning about Safety and Control in REGULA. In L. Sonenberg, P. Stone, K. Tumer, and P. Yolum, editors, *AAMAS 2011*, vol. 1–3, pp. 467–474, Taipei, Taiwan, May 2011. IFAAMAS.
14. M. Montali, P. Torroni, F. Chesani, P. Mello, M. Alberti, and E. Lamma. Abductive Logic Programming as an Effective Technology for the Static Verification of Declarative Business Processes. *Fundam. Inform.*, 102(3-4):325–361, 2010.
15. J.R. Searle. *The construction of social reality.* Free Press, New York, 1995.
16. M. P. Singh. An Ontology for Commitments in Multiagent Systems. *Artificial Intelligence and Law*, 7(1):97–113, 1999.
17. M. P. Singh. Formalizing Communication Protocols for Multiagent Systems. In M. Veloso, editor, *IJCAI 2007*, pp. 1519–1524, 2007. AAAI Press.
18. G. Weiss, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence.* The MIT Press, 1999.
19. M. Winikoff, W. Liu, and J. Harland. Enhancing Commitment Machines. In J. A. Leite, A. Omicini, P. Torroni, and P. Yolum, editors, *Proc. of DALT 2004*, LNCS 3476, pp. 198–220, July 2005. Springer.
20. P. Yolum and M. P. Singh. Designing and Executing Protocols Using the Event Calculus. In *Agents*, pp. 27–28, New York, NY, USA, 2001. ACM.
21. P. Yolum and M. P. Singh. Commitment Machines. In J.-J. Ch. Meyer and M. Tambe, editors, *Proc. of ATAL 2001*, LNCS 2333, pp. 235–247, 2002. Springer.