Timed Automata Semantics for Visual e-Contracts*

Enrique Martínez, M. Emilia Cambronero, Gregorio Díaz
Department of Computer Science
University of Castilla-La Mancha
Albacete, Spain
{emartinez, gregorio, emicp}@dsi.uclm.es

Gerardo Schneider
Department of Computer Science and Engineering
Chalmers | University of Gothenburg, Sweden
Department of Informatics
University of Oslo, Norway
gersch@chalmers.se

C-O Diagrams have been introduced as a means to have a more visual representation of electronic contracts, where it is possible to represent the obligations, permissions and prohibitions of the different signatories, as well as what are the penalties in case of not fulfillment of their obligations and prohibitions. In such diagrams we are also able to represent absolute and relative timing constraints. In this paper we present a formal semantics for C-O Diagrams based on timed automata extended with an ordering of states and edges in order to represent different deontic modalities.

1 Introduction

In the software context, the term *contract* has traditionally been used as a metaphor to represent limited kinds of "agreements" between software elements at different levels of abstraction. The first use of the term in connection with software programming and design was done by Meyer in the context of the language Eiffel (*programming-by-contracts*, or *design-by-contract*) [10]. This notion of contracts basically relies on the Hoare's notion of pre and post-conditions and invariants. Though this paradigm has proved to be useful for developing object oriented systems, it seems to have shortcomings for novel development paradigms such as service-oriented computing and component-based development. These new applications have a more involved interaction and therefore require a more sophisticated notion of contracts.

As a response, behavioural interfaces have been proposed to capture richer properties than simple pre and post-conditions [5]. Here it is possible to express contracts on the history of events, including causality properties. However, the approach is limited when it comes to contracts containing exceptional behaviour, since the focus is mainly on the interaction concerning expected (and prohibited) behaviour.

In the context of SOA, there are different service contract specification languages, like ebXML [4], WSLA [14], and WS-Agreement [13]. These standards and specification languages suffer from one or more of the following problems: They are restricted to bilateral contracts, lack formal semantics (so it is difficult to reason about them), their treatment of functional behaviour is rather limited and the sub-languages used to specify, for instance, security constraints are usually limited to small application-specific domains. The lack of suitable languages for contracts in the context of SOA is a clear conclusion of the survey [11] where a taxonomy is presented.

*Partially supported by the Spanish government (co financed by FEDER founds) with the project TIN2009-14312-C02-02 and the JCCLM regional project PEII09-0232-7745.

E. Pimentel, V. Valero (Eds.): Workshop on Formal Languages
and Analysis of Contract-Oriented Software 2011 (FLACOS'11)
EPTCS 68, 2011, pp. 7–21, doi:10.4204/EPTCS.68.3

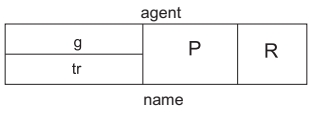

Figure 1: Box structure

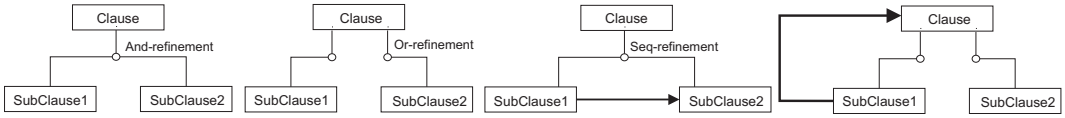

Figure 2: AND/OR/SEQ refinements and repetition in *C-O Diagrams*

More recently, some researchers have investigated how to adapt deontic logic [9] to define (consistent) contracts targeted to software systems where the focus is on the normative notions of obligation, permission and prohibition, including sometimes exceptional cases (e.g., [12]). Independently of the application domain, there still is need to better fill the gap between a contract understood by non-experts in formal methods (for its use), its logical representation (for reasoning), and its internal machine-representation (for runtime monitoring, and to be manipulated by programmers). We see two possible ways to bridge this gap: i) to develop suitable techniques to get a good translation from contracts written in natural language into formal languages, and ii) to provide a graphical representation (and tools) to manipulate contracts at a high level, with formal semantics supporting automatic translation into the formal language. We take in this paper the second approach.

In [8] we have introduced *C-O Diagrams*, a graphical representation for contracts allowing the representation of complex clauses describing the obligations, permissions, and prohibitions of different signatories (as defined in deontic logic [9]), as well as *reparations* describing contractual clauses in case of not fulfillment of obligations and prohibitions. Besides, *C-O Diagrams* permit to define real-time constraints. In [7] some of the satisfaction rules needed to check if a timed automaton satisfies a *C-O Diagram* specification were defined. These rules were originally miscalled "formal semantics". The goal of this paper is to further develop our previous work, in particular we present here a formal semantics for *C-O Diagrams* based on timed automata, extended with an ordering of states and edges.

The rest of the work is structured as follows: Section 2 presents *C-O Diagrams* and their syntax, Section 3 develops the formal semantics of *C-O Diagrams*, including its implementation in UPPAAL [6] and a small example. The work is concluded in Section 4.

2 C-O Diagrams Description and Syntax

In Fig. 1 we show the basic element of *C-O Diagrams*. It is called a **box** and it is divided into four fields. On the left-hand side of the box we specify the conditions and restrictions. The *guard* **g** specifies the conditions under which the contract clause must be taken into account (boolean expression). The *time restriction* **tr** specifies the time frame during which the contract clause must be satisfied (deadlines, timeouts, etc.). The *propositional content* **P**, on the center, is the main field of the box, and it is used to specify normative aspects (obligations, permissions and prohibitions) that are applied over actions, and/or the specification of the actions themselves. The last field of these boxes, on the right-hand side, is the *reparation* **R**. This reparation, if specified by the contract clause, is a reference to another contract that must be satisfied in case the main norm is not satisfied (a *prohibition* is violated or an *obligation* is not fulfilled, there is no reparation for *permission*), considering the clause eventually satisfied if this reparation is satisfied. Each box has also a name and an agent. The *name* is useful both to describe the clause and to reference the box from other clauses, so it must be unique. The *agent* indicates who is the performer of the action.

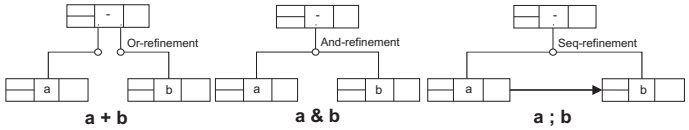

Figure 3: Compound actions in *C-O Diagrams*

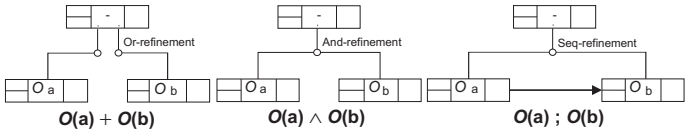

Figure 4: Composition of norms in *C-O Diagrams*

These basic elements of *C-O Diagrams* can be refined by using AND/OR/SEQ refinements, as shown in Fig. 2. The aim of these refinements is to capture the hierarchical clause structure followed by most contracts. An **AND-refinement** means that all the subclauses must be satisfied in order to satisfied the parent clause. An **OR-refinement** means that it is only necessary to satisfy one of the subclauses in order to satisfy the parent clause, so as soon as one of its subclauses is fulfilled, we conclude that the parent clause is fulfilled as well. A **SEQ-refinement** means that the norm specified in the target box (*SubClause2* in Fig. 2) must be fulfilled after satisfying the norm specified in the source box (*SubClause1* in Fig. 2). By using these structures we can build a hierarchical tree with the clauses defined by a contract, where the leaf clauses correspond to the atomic clauses, that is, to the clauses that cannot be divided into subclauses. There is another structure that can be used to model **repetition**. This structure is represented as an arrow going from a subclause to one of its ancestor clauses (or to itself), meaning the repetitive application of all the subclauses of the target clause after satisfying the source subclause. For example, in the right-hand side of Fig. 2, we have an **OR-refinement** with an arrow going from *SubClause1* to *Clause*. It means that after satisfying *SubClause1* we apply *Clause* again, but not after satisfying *SubClause2*.

It is only considered the specification of *atomic actions* in the **P** field of the leaf boxes of our diagrams. The composition of actions can be achieved by means of the different kinds of refinement. In this way, an AND-refinement can be used to model *concurrency* "&" between actions, an OR-refinement can be used to model a *choice* "+" between actions, and a SEQ-refinement can be used to model *sequence* ";" of actions. In Fig. 3 we can see an example about how to model these compound actions through refinements, given two atomic actions *a* and *b*.

The *deontic norms* (obligations, permissions and prohibitions) that are applied over these actions can be specified in any box of our *C-O Diagrams*, affecting all the actions in the leaf boxes that are descendants of this box. If it is the case that the box where we specify the deontic norm is a leaf, the norm only affects the atomic action we have in this box. It is used an upper case "*O*" to denote an obligation, an upper case "*P*" to denote a permission, and an upper case "*F*" to denote a prohibition (forbidden). These letters are written in the top left corner of field **P**.

The composition of deontic norms is also achieved by means of the different refinements we have in *C-O Diagrams*. Thus, an AND-refinement corresponds to the *conjunction* operator "$\wedge$" between norms, an OR-refinement corresponds to the *choice* operator "$+$" between norms, and a SEQ-refinement corresponds to the *sequence* operator ";" between norms. For example, we can imagine having a leaf box specifying the obligation of performing an action *a*, written as O(a), and another leaf box specifying the obligation of performing an action *b*, written as O(b). These two norms can be combined in the three different ways mentioned before through the different kinds of refinement (Fig. 4). However, the specification of deontic norms in our diagrams must fulfill the following rule: exactly one deontic norm must be specified in each one of the branches of our hierarchical tree, i.e., we cannot have an action without a deontic norm applied over it and we cannot have deontic norms applied over other deontic

norms. We have also that *agents* are only specified in the boxes where a deontic norm is defined, being each agent associated to a concrete deontic norm. Finally, the *repetition* of both, actions and deontic norms, can be achieved by means of the repetition structure we have in *C-O Diagrams*.

We have given here an abridged description of *C-O Diagrams*. A more detail description can be found in [8], including a qualitative and quantitative evaluation, and a discussion on related work.

Definition 1 *(C-O Diagrams Syntax) We consider a finite set of real-valued variables $\mathscr{C}$ standing for clocks, a finite set of non-negative integer-valued variables $\mathscr{V}$, a finite alphabet Σ for atomic actions, a finite set of identifiers $\mathscr{A}$ for agents, and another finite set of identifiers $\mathscr{N}$ for names. The greek letter ε means that and expression is not given, i.e., it is empty.*

We use C to denote the contract modelled by a C-O Diagram. The diagram is defined by the following EBNF grammar:

$$
\begin{array}{rcl}
C & := & (agent, name, g, tr, O(C_2), R) \,| \\
 & & (agent, name, g, tr, P(C_2), \varepsilon) \,| \\
 & & (agent, name, g, tr, F(C_2), R) \,| \\
 & & (\varepsilon, name, g, tr, C_1, \varepsilon) \\
C_1 & := & C\,(And\ C)^+ \,|\, C\,(Or\ C)^+ \,|\, C\,(Seq\ C)^+ \\
C_2 & := & a \,|\, C_3\,(And\ C_3)^+ \,|\, C_3\,(Or\ C_3)^+ \,|\, C_3\,(Seq\ C_3)^+ \\
C_3 & := & (\varepsilon, name, \varepsilon, \varepsilon, C_2, \varepsilon) \\
R & := & C \,|\, \varepsilon
\end{array}
$$

where $a \in \Sigma$, $agent \in \mathscr{A}$ and $name \in \mathscr{N}$. Guard g is ε or a conjunctive formula of atomic constraints of the form: $v \sim n$ or $v - w \sim n$, for $v, w \in \mathscr{V}$, $\sim \in \{\leq, <, =, >, \geq\}$ and $n \in \mathbb{N}$, whereas timed restriction tr is ε or a conjunctive formula of atomic constraints of the form: $x \sim n$ or $x - y \sim n$, for $x, y \in \mathscr{C}$, $\sim \in \{\leq, <, =, >, \geq\}$ and $n \in \mathbb{N}$. O, P and F are the deontic operators corresponding to obligation, permission and prohibition, respectively, where $O(C_2)$ states the obligation of performing C_2, $F(C_2)$ states prohibition of performing C_2, and $P(C_2)$ states the permission of performing C_2. And, Or and Seq are the operators corresponding to the refinements we have in C-O Diagrams, AND-refinement, OR-refinement and SEQ-refinement, respectively.

The simplest contract we can have in *C-O Diagrams* is that composed of only one box including the elements *agent* and *name*. Optionally, we can specify a guard g and a time restriction tr. We also have a deontic operator (O, P or F) applied over an atomic action a, and in the case of obligations and prohibitions it is possible to specify another contract C as a reparation.

We use C_1 to define a more complex contract where we combine different deontic norms by means of any of the different refinements we have in *C-O Diagrams*. In the box where we have the refinement into C_1 we cannot specify an agent nor a reparation because these elements are always related to a single deontic norm, but we still can specify a guard g and a time restriction tr that affect all the deontic norms we combine.

Once we write a deontic operator in a box of our diagram, we have two possibilities as we can see in the specification of C_2: we can just write a simple action a in the box, being the deontic operator applied only over it, or we can refine this box in order to apply the deontic operator over a compound action. In this case we have that the subboxes (C_3) cannot define a new deontic operator as it has already been defined in the parent box (affecting all the subboxes).

3 *C-O Diagrams* Semantics

The *C-O Diagrams* semantics is defined by means of a transformation into a *Network of Timed Automata* (NTA), that is defined as a set of timed automata [1, 2] that run simultaneously, using the same set of clocks and variables, and synchronizing on the common actions.

In what follows we consider a finite set of real-valued variables $\mathscr{C}$ ranged over by $x,y,\ldots$ standing for clocks, a finite set of non-negative integer-valued variables $\mathscr{V}$, ranged over by $v,w,\ldots$ and a finite alphabet Σ ranged over by $a,b,\ldots$ standing for actions. We will use letters $r,r',\ldots$ to denote sets of clocks. We will denote by *Assigns* the set of possible assignments, $Assigns = \{v := expr \mid v \in \mathscr{V}\}$, where *expr* are arithmetic expressions using naturals and variables. Letters $s,s'\ldots$ will be used to represent a set of assignments.

A *guard or invariant condition* is a conjunctive formula of atomic constraints of the form: $x \sim n$, $x-y \sim n$, $v \sim n$ or $v-w \sim n$, for $x,y \in \mathscr{C}$, $v,w \in \mathscr{V}$, $\sim \in \{\leq,<,=,>,\geq\}$ and $n \in \mathbb{N}$. The set of guard or invariant conditions will be denoted by $\mathscr{G}$, ranged over by $g,g',\ldots$.

Definition 2 *(Timed Automaton)*
A timed automaton *is a tuple* (N,n_0,E,I)*, where N is a finite set of locations (nodes),* $n_0 \in N$ *is the initial location,* $E \subseteq N \times \mathscr{G} \times \Sigma \times \mathscr{P}(Assigns) \times 2^{\mathscr{C}} \times N$ *is the set of edges, where the subset of* urgent edges *is called* $E_u \subseteq E$*, and they will graphically be distinguished as they will have their arrowhead painted in white.* $I : N \to \mathscr{G}$ *is a function that assigns invariant conditions (which could be empty) to locations.*

From now on, we will write $n \xrightarrow{g,a,r}_{s} n'$ to denote $(n,g,a,s,r,n') \in E$, and $n \xrightarrow{g,a,r}_{s} {}_{u}\ n'$ when $(n,g,a,s,r,n') \in E_u$.

In an NTA we distinguish two types of actions: internal and synchronization actions. Internal actions can be executed by the corresponding automata independently, and they will be ranged over the letters $a,b\ldots$. Synchronization actions, however, must be executed simultaneously by two automata, and they will be ranged over letters $m,m',\ldots$ and come from the synchronization of two actions $m!$ and $m?$, executed from two different automata. Due to the lack of space, we refer the reader to [3] for the definition of the semantics of timed automaton and NTA.

To specify the *C-O Diagrams* semantics, we add the definition of two orderings, $\prec_N$ and $\prec_E$, where:

- $\prec_N$ is a (strict, partial) ordering on N where $n \prec_N n'$ means that node n is *better* than node n'.
- $\prec_E$ is a (strict, partial) ordering on E where $e \prec_N e'$ means that edge e is *better* than edge e'.

We also add a **violation set** $V(n)$ associated to each node n in N, that is the set of contractual obligations and prohibitions that are violated in n.

Definition 3 *(Violation Set) Let us consider the set of contractual obligations and prohibitions CN ranged over cn,* $cn',\ldots$ *standing for identifiers of obligations and prohibitions. We write* $n \not\models cn$ *to express that obligation or prohibition cn is violated in node n. Therefore, the* violation set *is defined as* $V(n) = \{cn \mid cn \in CN \text{ and } n \not\models cn\}$.

Another set called **satisfaction set** $S(n)$ is also associated to each node n in N. This set is composed by the contractual obligations and prohibitions that have already been satisfied in n.

Definition 4 *(Satisfaction Set) Let us consider the set of contractual obligations and prohibitions COF ranged over* cof, $cof',\ldots$ *standing for identifiers of obligations and prohibitions. We write* $n \models cof$ *to express that obligation or prohibition* cof *has been satisfied in node n (we consider a prohibition satisfied in node n if it has not been violated and cannot be violated anymore because the time frame specified for the prohibition has expired). Hence, the* satisfaction set *is defined as* $S(n) = \{cof \mid cof \in COF \text{ and } n \models cof\}$.

Once these two sets have been defined, we can formally define the **ordering on nodes** $\prec_N$, by comparing the violation sets and the satisfaction sets of the nodes, and the **ordering on edges** $\prec_E$, by comparing the violation sets and the satisfaction sets of the target nodes of the edges.

Definition 5 *(Ordering on Nodes) A* node n_1 is better than another node n_2 *if the violation set of* n_1 *is a proper subset of the violation set of* n_2 *or, if the violation sets are the same, a node* n_1 *is better than another node* n_2 *if the satisfaction set of* n_1 *is a proper superset of the satisfaction set of* n_2*, that is,* $n_1 \prec_N n_2$ *iff* $(V(n_1) \subset V(n_2))$ *or* $(V(n_1) = V(n_2)$ *and* $S(n_1) \supset S(n_2))$.

Definition 6 *(Ordering on Edges) An* edge e_1 is better than another edge e_2 *if the source node is the same in both cases but the violation set of the target node of* e_1 *is a proper subset of the violation set of the target node of* e_2 *or, if the violation sets are the same, an edge* e_1 *is better than another edge* e_2 *if the satisfaction set of the target node of* e_1 *is a proper superset of the satisfaction set of the target node of* e_2*. Considering* $e_1 = (n_1, g_1, a_1, s_1, r_1, n_1')$ *and* $e_2 = (n_2, g_2, a_2, s_2, r_2, n_2')$*,* $e_1 \prec_E e_2$ *iff* $(n_1 = n_2)$ *and* $(V(n_1') \subset V(n_2')$ *or* $(V(n_1') = V(n_2')$ *and* $S(n_1') \supset S(n_2')))$.

Finally, another set called **permission set** $P(n)$ is associated to each node n in N. This set influences neither the ordering on nodes nor the ordering on edges, it is used just to record the permissions in the contract that have been made effective.

Definition 7 *(Permission Set) Let us consider the set of contractual permissions CP ranged over* cp*,* cp'*,... standing for identifiers of permissions. We write* $n \models cp$ *to express that permission* cp *has already been made effective in node n. Then, the* permission set *is defined as* $P(n) = \{cp \mid cp \in CP$ *and* $n \models cp\}$.

Graphically, when we draw a timed automaton extended with these three sets, we write under each node n between braces its violation set $V(n)$ on the left, its satisfaction set $S(n)$ on the centre and its permission set $P(n)$ on the right. In the initial node of the automata we build corresponding to *C-O Diagrams* these three sets are empty. By default, a node keeps in these sets the same content of the previous node when we compose the automata. Only in a few cases the content of these sets is modified (when an obligation or a prohibition is violated, an obligation or a prohibition is satisfied or a permission is made effective).

Concerning the **real-time restrictions** tr specified in the contract, the two types of time restrictions we can have in *C-O Diagrams* must be translated in a different way for their inclusion into a timed automaton construction:

- A time restriction specified using **absolute time** must be specified in timed automata by rewriting the terms in which absolute time references occur. For that purpose we define a global clock $T \in \mathscr{C}$ that is never reset during the execution of the automata and, taking into account the moment at which the contract is enacted, we rewrite the absolute time references as deadlines involving clock T and considering the smallest time unit needed in the contract. For example, let us consider a clause that must be satisfied between the *5th of November* and the *10th of November*, and that the contract containing this clause is enacted the *31st of October*. If we suppose that *days* is the smallest time unit used in the contract for the specification of real-time restrictions, the time restriction of this clause is written as $(T \geq 5)$ *and* $(T \leq 10)$.
- A time restriction specified using **relative time** must be specified in timed automata by introducing an additional clock to register the amount of time that has elapsed since another clause has been satisfied, resetting the additional clock value when this happens and specifying the deadline using it. We call this clock t_{name}, where *name* is the clause used as reference for the specification of the time restriction. Therefore, we define a set of additional clocks $C_{add} = \{t_{name} \mid t_{name} \in \mathscr{C}\}$ including

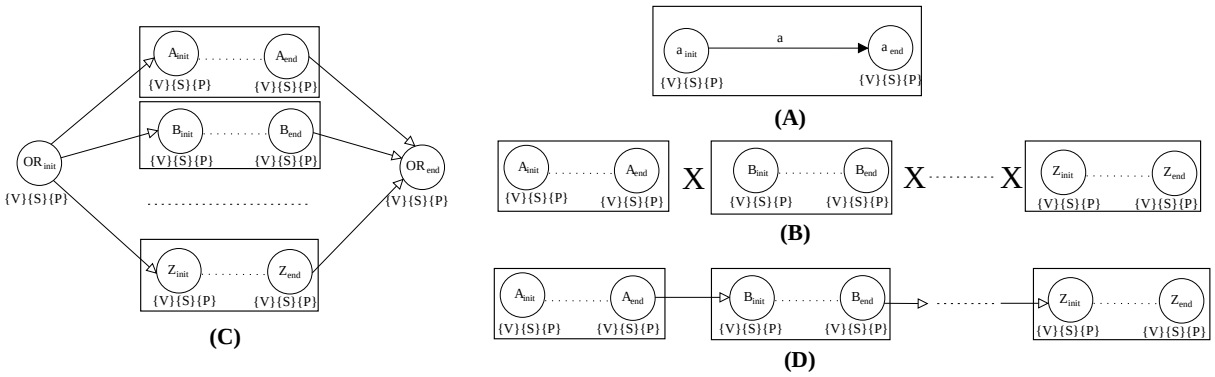

Figure 5: Automata corresponding to a **simple action** *a* and to **compound actions**

a clock for every clause that is used as reference in the time restriction of at least another clause. For example, let us consider a contract with a clause that must be satisfied between *5* and *10* days after another clause *name*1 has been satisfied. In this case we define an additional clock t_{name1} that is reset to zero when clause *name*1 is satisfied ($t_{name1} := 0$) and the time restriction of the other clause is written as $(t_{name1} \geq 5)\, and\, (t_{name1} \leq 10)$.

As a result, the set of clocks of the timed automata would be $\mathscr{C} = \{T\} \cup C_{add}$. When we construct the timed automata corresponding to *C-O Diagrams*, we always consider $(x \geq t1)\, and\, (x \leq t2)$ as the interval corresponding to the time restriction tr of the clause, where $x \in \mathscr{C}$ is the clock used for its specification ($x = T$ in the case of absolute time and $x = t_{name}$ in the case of relative time, being *name* the clause used as reference), $t1 \in \mathbb{N}$ is the beginning of the interval and $t2 \in \mathbb{N}$ is the end of the interval ($t1 \leq t2$). If tr does not define the lower bound of the interval we take $t1 = 0$, if tr does not define the upper bound of the interval we take $t2 = \infty$, and if $tr = \varepsilon$ we take $t1 = 0$, $t2 = \infty$ and $x = T$.

Once we have given these extensions of the definition of timed automata and we have explained how the different kinds of time restriction can be expressed, considering all the different elements we can specify in a *C-O Diagram*, we can define the transformation of the diagrams into timed automata by induction using several transformation rules.

Definition 8 *(C-O Diagrams Transformation Rules: Part I)*

*(1) An **atomic action** in a C-O Diagram, that is, $(\varepsilon, name, \varepsilon, \varepsilon, a, \varepsilon)$ corresponds to the timed automaton $A = (N_A, n_{0_A}, E_A, I_A)$, where:*

- $N_A = \{a_{init}, a_{end}\}$.
- $n_{0_A} = a_{init}$.
- $E_A = \{a_{init} \xrightarrow{a} a_{end}\}$.
- $I_A = \emptyset$.

*The violation **(V)**, satisfaction **(S)** and permission **(P)** sets are not modified, so $V(a_{init}) = V(a_{end})$, $S(a_{init}) = S(a_{end})$ and $P(a_{init}) = P(a_{end})$. This timed automaton can be seen in Fig. 5 **(A)**.*

*(2) A **compound action** in a C-O Diagram where an **AND-refinement** is used to compose actions, that is, $(\varepsilon, name, \varepsilon, \varepsilon, C_1 And C_2 And \ldots And C_n, \varepsilon)$ corresponds to the cartesian product of the automata corresponding to each one of the subcontracts. Let us consider $A, B, \ldots, Z$ the automata corresponding to the subcontracts $C_1, C_2, \ldots, C_n$ (the actions specified in these subcontracts can be atomic actions or other compound actions). The resulting automaton AND corresponds to the cartesian product of these automata, that is, $AND = A \times B \times \ldots \times Z$. Again, the violation **(V)**, satisfaction **(S)** and permission **(P)** sets are not modified, so they are the same in all the nodes. This composition of timed automata is shown graphically in Fig. 5 **(B)**.*

(3) *A **compound action** in a C-O Diagram where an **OR-refinement** is used to compose actions, that is, $(\varepsilon, name, \varepsilon, \varepsilon, C_1 Or C_2 Or \ldots Or C_n, \varepsilon)$ corresponds to a new automaton in which the automata corresponding to each one of the subcontracts is considered as an alternative. Let us consider $A, B, \ldots, Z$ the automata corresponding to the subcontracts $C_1, C_2, \ldots, C_n$ (the actions specified in these subcontracts can be atomic actions or other compound actions). The resulting automaton OR preserves the structure of the automata we are composing but adding a new initial node OR_{init} and connecting this node by means of urgent edges performing no action to the initial nodes of $A, B, \ldots, Z$ ($A_{init}, B_{init}, \ldots, Z_{init}$). It is also added a new ending node OR_{end} and urgent edges performing no action from the ending nodes of $A, B, \ldots, Z$ ($A_{end}, B_{end}, \ldots, Z_{end}$) to this new ending node. Let $A = (N_A, n0_A, E_A, I_A), B = (N_{\mathscr{B}}, n0_B, E_B, I_B), \ldots, Z = (N_Z, n0_Z, E_Z, I_Z)$. The resulting automaton is therefore $OR = (N_{OR}, n0_{OR}, E_{OR}, I_{OR})$, where:*

- $N_{OR} = N_A \cup N_B \cup \ldots \cup N_Z \cup \{OR_{init}, OR_{end}\}$.
- $n0_{OR} = OR_{init}$.
- $E_{OR} = E_A \cup E_B \cup \ldots \cup E_Z \cup \{OR_{init} \longrightarrow_u A_{init}, OR_{init} \longrightarrow_u B_{init}, \ldots, OR_{init} \longrightarrow_u Z_{init}\} \cup \{A_{end} \longrightarrow_u OR_{end}, B_{end} \longrightarrow_u OR_{end}, \ldots, Z_{end} \longrightarrow_u OR_{end}\}$.
- $I_{OR} = I_A \cup I_B \cup \ldots \cup I_Z$.

*The violation (**V**), satisfaction (**S**) and permission (**P**) sets are not modified, so they are the same in all the nodes. This composition of timed automata is shown graphically in Fig. 5 **(C)**.*

(4) *A **compound action** in a C-O Diagram where a **SEQ-refinement** is used to compose actions, that is, $(\varepsilon, name, \varepsilon, \varepsilon, C_1 Seq C_2 Seq \ldots Seq C_n, \varepsilon)$ corresponds to a new automaton in which the automata corresponding to each one of the subcontracts are connected in sequence. Let us consider $A, B, \ldots, Z$ the automata corresponding to the subcontracts $C_1, C_2, \ldots, C_n$ (the actions specified in these subcontracts can be atomic actions or other compound actions). The resulting automaton SEQ preserves the structure of the automata we are composing, adding no extra nodes. We only connect with an urgent edge performing no action the ending node of each automaton in the sequence ($A_{end}, B_{end}, \ldots, Y_{end}$) with the initial node of the next automaton in the sequence ($B_{init}, C_{init}, \ldots, Z_{init}$). This rule is not applied in the cases of A_{init} (as there is not previous ending node to connect) and Z_{end} (as there is not following initial node to connect). Let $A = (N_A, n0_A, E_A, I_A), \mathscr{B} = (N_B, n0_B, E_B, I_B), \ldots, Z = (N_Z, n0_Z, E_Z, I_Z)$. The resulting automaton is therefore $SEQ = (N_{SEQ}, n0_{SEQ}, E_{SEQ}, I_{SEQ})$, where:*

- $N_{SEQ} = N_A \cup N_B \cup \ldots \cup N_Z$.
- $n0_{SEQ} = A_{init}$.
- $E_{SEQ} = E_A \cup E_B \cup \ldots \cup E_Z \cup \{A_{end} \longrightarrow_u B_{init}, B_{end} \longrightarrow_u C_{init}, \ldots, Y_{end} \longrightarrow_u Z_{init}\}$.
- $I_{SEQ} = I_A \cup I_B \cup \ldots \cup I_Z$.

*Again, the violation (**V**), satisfaction (**S**) and permission (**P**) sets are not modified, so they are the same in all the nodes. This composition of timed automata is shown graphically in Fig. 5 **(D)**.*

Until now, we have seen how the automata corresponding to the different actions (atomic or compound) specified in a *C-O Diagram* are constructed and we have seen that these translations do not modify the content of any of the sets (violation, satisfaction or permission). Next, we define the transformation rules specifying how these "action" automata are modified when we apply a deontic norm (obligation, permission or prohibition) over the actions in the *C-O Diagram*.

Definition 7 *(C-O Diagrams Transformation Rules: Part II)*

(5) *The application of an* ***obligation****, a* ***permission*** *or a* ***prohibition*** *over an action in a C-O Diagram, i.e.,* $(agent, name, g, tr, O/P/F(C), R)$ *corresponds to an automaton where the obligation/prohibition of performing the action specified in the subcontract C can be skipped, fulfilled or violated, whereas the permission of performing the action can be skipped, made effective or not made effective. Let us consider* $A = (N_A, n_{0_A}, E_A, I_A)$ *the automaton corresponding to C, being* A_{init} *the initial node and* A_{end} *the ending node. The resulting automaton* $D(A)$*, where* $D \in \{O, P, F\}$*, preserves the structure of the automaton A but adding a new ending node* A_{time} *including the obligation over the action in its violation set, the prohibition over the action in its satisfaction set or nothing if a permission over the action is considered. If guard condition* $g \neq \varepsilon$*, we add another ending node* A_{skip} *where the violation, satisfaction and permission sets are not modified. We also include the obligation over the action in the satisfaction set of* A_{end}*, the prohibition over the action in the violation set of* A_{end}*, or the permission over the action in the permission set of* A_{end}*. An invariant* $x \leq t2+1$ *is added to each node of A except* A_{end} *and each edge performing one of the actions in this automaton is guarded with* $(x \geq t1)$ *and* $(x \leq t2)$ *and action performed by agent. New edges guarded with* $x = t2+1$ *and no action performed are added from each node of A except* A_{end} *to the new node* A_{time} *and, if guard condition* $g \neq \varepsilon$*, an urgent edge from* A_{init} *to* A_{skip} *is also added guarded with the guard condition of the clause negated* $(\neg g)$*. Finally, if* $t_{name} \in \mathscr{C}$*, all the edges reaching* A_{end} *reset* t_{name} *in the cases of obligation and permission, whereas all the edges reaching* A_{time} *reset* t_{name} *in the case of prohibition. Considering the more complex case, where* $g \neq \varepsilon$ *and* $t_{name} \in \mathscr{C}$*, and having that* $g_1 \equiv (x \geq t1)$ *and* $(x \leq t2)$ *and* $g_2 \equiv x = t2+1$*, the resulting automaton is therefore* $D(A) = (N_{D(A)}, n_{0_{D(A)}}, E_{D(A)}, I_{D(A)})$*, where:*

- $N_{D(A)} = N_A \cup \{A_{time}, A_{skip}\}$.
- $n_{0_{D(A)}} = A_{init}$.
- $E_{D(A)} = \{A_{init} \xrightarrow{\neg g}_u A_{skip}\} \cup$
$$\begin{cases} n \xrightarrow{g_1, agent(a)} n' \,|\, n \xrightarrow{a} n' \in E_A \text{ and } n' \neq A_{end}, \\ n \xrightarrow{g_1, agent(a), t_{name}} n' \,|\, n \xrightarrow{a} n' \in E_A \text{ and } n' = A_{end}, \\ n \xrightarrow{g_2} A_{time} \,|\, n \in N_A - \{A_{end}\} & \text{if } D = O \\ n \xrightarrow{g_1, agent(a)} n' \,|\, n \xrightarrow{a} n' \in E_A \text{ and } n' \neq A_{end}, \\ n \xrightarrow{g_1, agent(a), t_{name}} n' \,|\, n \xrightarrow{a} n' \in E_A \text{ and } n' = A_{end}, \\ n \xrightarrow{g_2} A_{time} \,|\, n \in N_A - \{A_{end}\} & \text{if } D = P \\ n \xrightarrow{g_1, agent(a)} n' \,|\, n \xrightarrow{a} n' \in E_A, \\ n \xrightarrow{g_2, t_{name}} A_{time} \,|\, n \in N_A - \{A_{end}\} & \text{if } D = F \end{cases}$$
- $I_{D(A)} = I_A \cup \{I(n) \equiv x \leq t2+1 \,|\, n \in N_A - \{A_{end}\}\}$.

The resulting timed automata are shown graphically in Fig. 6, where ***(A)*** *corresponds to obligation,* ***(B)*** *corresponds to permission and* ***(C)*** *corresponds to prohibition. We consider a one of the atomic actions included in the subcontract C.*

We can see that the above constructions can include a reparation contract *R* in the cases of obligation and prohibition. If this reparation is defined, we have to construct the automaton corresponding to the reparation contract and integrate this automaton as part of the automaton we have generated for the obligation or prohibition. This reparation contract removes the obliged or prohibited clause *name* from the violation set of the corresponding automaton, as we can see in Fig. 6 **(D)**.

Definition 7 *(C-O Diagrams Transformation Rules: Part III)*

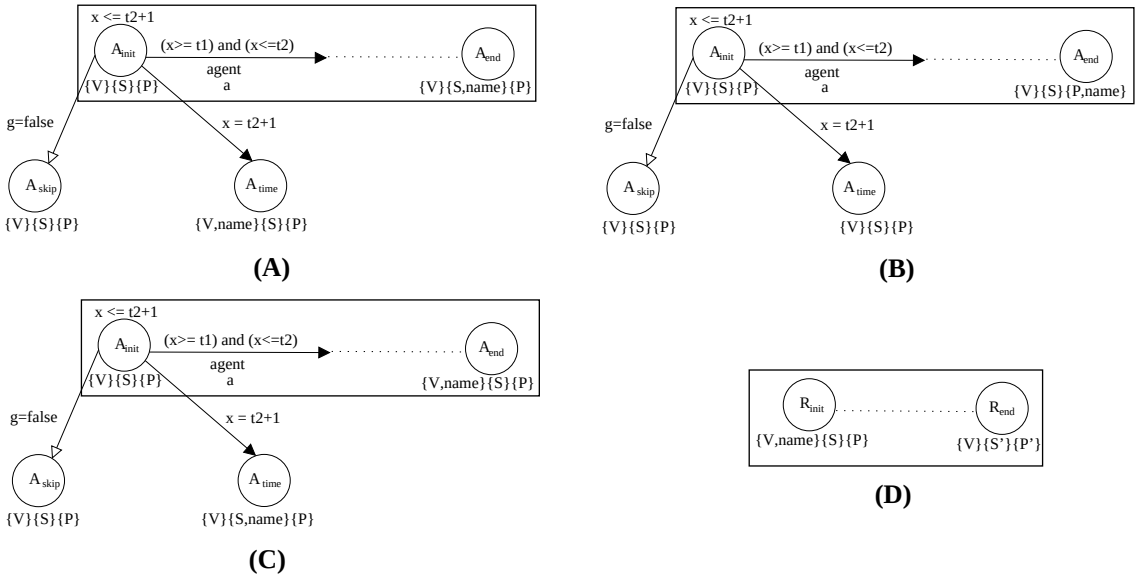

Figure 6: Automata corresponding to **deontic norms** and automaton corresponding to a **reparation**

*(6) An **obligation** or **prohibition** in a C-O Diagram specifying a contract **reparation** $R \neq \varepsilon$ corresponds to the obligation automaton $O(A)$ or the prohibition automaton $F(A)$ together with the reparation automaton R, considering the node with name in its violation (A_{vio}) set as the initial node of the reparation automaton (R_{init}). In the ending node of the reparation automaton (R_{end}) name is removed from the violation set, as the violation has been repaired. In this node we also have that the satisfaction set and the permission set are different from the ones we have in the initial node of the reparation because we have to include in the satisfaction set all the obligations and prohibitions satisfied in the reparation contract, and in the permission set all the permissions that have been made effective in the reparation contract. Let us consider $D(A) = (N_{D(A)}, n0_{D(A)}, E_{D(A)}, I_{D(A)})$, where $D \in \{O,F\}$, and $R = (N_R, n0_R, E_R, I_R)$. The resulting automaton is therefore $D(A)_R = (N_{D(A)_R}, n0_{D(A)_R}, E_{D(A)_R}, I_{D(A)_R})$, where:*

- $N_{D(A)_R} = N_{D(A)} \cup N_R - \{R_{init}\}$.
- $n0_{D(A)_R} = A_{init}$.
- $E_{D(A)_R} = E_{D(A)} \cup \{n \xrightarrow[s]{g,a,r} n' \mid n \xrightarrow[s]{g,a,r} n' \in E_R \text{ and } n \neq R_{init}\} \cup$
  $\{A_{vio} \xrightarrow[s]{g,a,r} n' \mid n \xrightarrow[s]{g,a,r} n' \in E_R \text{ and } n = R_{init}\}$.
- $I_{D(A)_R} = I_{D(A)} - \{I(A_{vio})\} \cup \{I(n) \mid n \in N_R - \{R_{init}\}\} \cup \{I(A_{vio}) \equiv I(R_{init})\}$.

Finally, we have to define the rules about how the automata corresponding to different deontic norms are composed when we have a composition of deontic norms in our *C-O Diagram*. To make this composition possible, first we need to have only one ending node in the automata corresponding to the different deontic norms. Therefore, we add a new ending node in these automata and urgent edges from the old ending nodes to this new node. Notice that in the case of obligation and prohibition, if there is no reparation defined, the node violating the norm is a final node of the whole automaton construction where the contract is breached. In the case of permission, as no reparation is defined, we have that $P(A)_R = P(A)$.

Definition 7 *(C-O Diagrams Transformation Rules: Part IV)*

*(7) Let $D(A)_R = (N_{D(A)_R}, n0_{D(A)_R}, E_{D(A)_R}, I_{D(A)_R})$, where $D \in \{O,P,F\}$, be the automaton corresponding to an **obligation**, a **prohibition** or a **permission** in a C-O Diagram, specifying a **reparation** $R \neq \varepsilon$ in the two first cases. The corresponding automaton with only one ending node, that we call A_{final}*

and preserves the violation, satisfaction and permission sets of the previous node, is $D(A)'_R = (N_{D(A)'_R}, n0_{D(A)'_R}, E_{D(A)'_R}, I_{D(A)'_R})$*, where:*

- $N_{D(A)'_R} = N_{D(A)_R} \cup \{A_{final}\}$.
- $n0_{D(A)'_R} = n0_{D(A)_R}$.
- $E_{D(A)'_R} = E_{D(A)_R} \cup \{A_{skip} \longrightarrow_u A_{final}\} \cup \begin{cases} A_{end} \longrightarrow_u A_{final}, R_{end} \longrightarrow_u A_{final} & \text{if } D = O \\ A_{end} \longrightarrow_u A_{final}, A_{time} \longrightarrow_u A_{final} & \text{if } D = P \\ A_{time} \longrightarrow_u A_{final}, R_{end} \longrightarrow_u A_{final} & \text{if } D = F \end{cases}$
- $I_{D(A)'_R} = I_{D(A)_R}$.

Therefore, the composition of the automata corresponding to different deontic norms is defined by three additional transformation rules.

Definition 7 *(C-O Diagrams Transformation Rules: Part V)*

(8) If several norms are composed by an ***AND-refinement****, that is, we have specified the diagram* $(\varepsilon, name, g, tr, C_1 And C_2 And \ldots And C_n, \varepsilon)$*, their composition corresponds to a network of automata in which we consider all the norms we are composing in parallel. Let us consider* $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ *the automata corresponding to the norms we are composing. The resulting network of automata preserves the structure of the automata we are composing, adding to each one of them the additional nodes and edges necessary for synchronization (these nodes are called* C_{init} *and* C_{final} *in the first automaton,* C_{isyn} *and* $C_{isyn'}, i = 1, \ldots, n-1$ *in the other automata). Before its initial node, each automaton synchronizes with the other automata and it synchronizes again after its final node by means of urgent channels* $(m_1, m_2, \ldots, m_{n-1})$*. In the first automaton we add another node* C_{skip} *if guard condition of the parent clause* $g \neq \varepsilon$ *and an urgent edge from* C_{init} *to this new node guarded with the guard condition negated* $(\neg g)$*. In the final node of the first automaton the violation, satisfaction and permission sets are the union of the sets resulting in each one of the automata running in parallel, so we have that Vfinal =* $V1 \cup V2 \cup \ldots \cup Vn$*, Sfinal =* $S1 \cup S2 \cup \ldots \cup Sn$ *and Pfinal =* $P1 \cup P2 \cup \ldots \cup Pn$*. If time restriction of the parent clause* $tr \neq \varepsilon$*, we consider this additional time restriction in all the composed automata together with their own time restrictions. Let* $\mathscr{C}_1 = (N_{\mathscr{C}_1}, n0_{\mathscr{C}_1}, E_{\mathscr{C}_1}, I_{\mathscr{C}_1}), \mathscr{C}_2 = (N_{\mathscr{C}_2}, n0_{\mathscr{C}_2}, E_{\mathscr{C}_2}, I_{\mathscr{C}_2}), \ldots, \mathscr{C}_n = (N_{\mathscr{C}_n}, n0_{\mathscr{C}_n}, E_{\mathscr{C}_n}, I_{\mathscr{C}_n})$*. Considering the case where* $g \neq \varepsilon$ *and* $tr \neq \varepsilon$*, and having that* $E_{\mathscr{C}_1}*, E_{\mathscr{C}_2}*, \ldots, E_{\mathscr{C}_n}*$ *are the sets of edges considering time restriction* tr *together with their own time restriction, the resulting network of automata is therefore* $\mathscr{C}*_i = (N_{\mathscr{C}*_i}, n0_{\mathscr{C}*_i}, E_{\mathscr{C}*_i}, I_{\mathscr{C}*_i})$*,* $i = 1, \ldots, n$ *where:*

- $N_{\mathscr{C}*_i} = N_{\mathscr{C}_i} \cup \begin{cases} C_{init}, C_{final}, C_{skip} & \text{if } i = 1 \\ C_{isyn}, C_{isyn'}, C_{i-1syn}, C_{i-1syn'} & \text{if } i = 2, \ldots, n-1 \\ C_{i-1syn}, C_{i-1syn'} & \text{if } i = n \end{cases}$
- $n0_{\mathscr{C}*_i} = \begin{cases} C_{init} & \text{if } i = 1 \\ C_{i-1syn}, C_{i-1syn'} & \text{if } i = 2, \ldots, n \end{cases}$
- $E_{\mathscr{C}*_i} = E_{\mathscr{C}_i} \cup \begin{cases} C_{init} \xrightarrow{\neg g}_u C_{skip}, C_{init} \xrightarrow{m_i!} C_{iinit}, & \\ C_{ifinal} \xrightarrow{m_i!} C_{final} & \text{if } i = 1 \\ C_{i-1syn} \xrightarrow{m_{i-1}?} C_{isyn}, C_{isyn'} \xrightarrow{m_{i-1}?} C_{i-1syn'}, & \\ C_{isyn} \xrightarrow{m_i!} C_{iinit}, C_{ifinal} \xrightarrow{m_i!} C_{isyn'} & \text{if } i = 2, \ldots, n-1 \\ C_{i-1syn} \xrightarrow{m_{i-1}?} C_{iinit}, C_{ifinal} \xrightarrow{m_{i-1}?} C_{final} & \text{if } i = n \end{cases}$
- $I_{\mathscr{C}*_i} = I_{\mathscr{C}_i} \cup \{I(n) \equiv x \leq t2 + 1 \,|\, n \in N_{\mathscr{C}_i} - \{C_{ifinal}\}\}$.

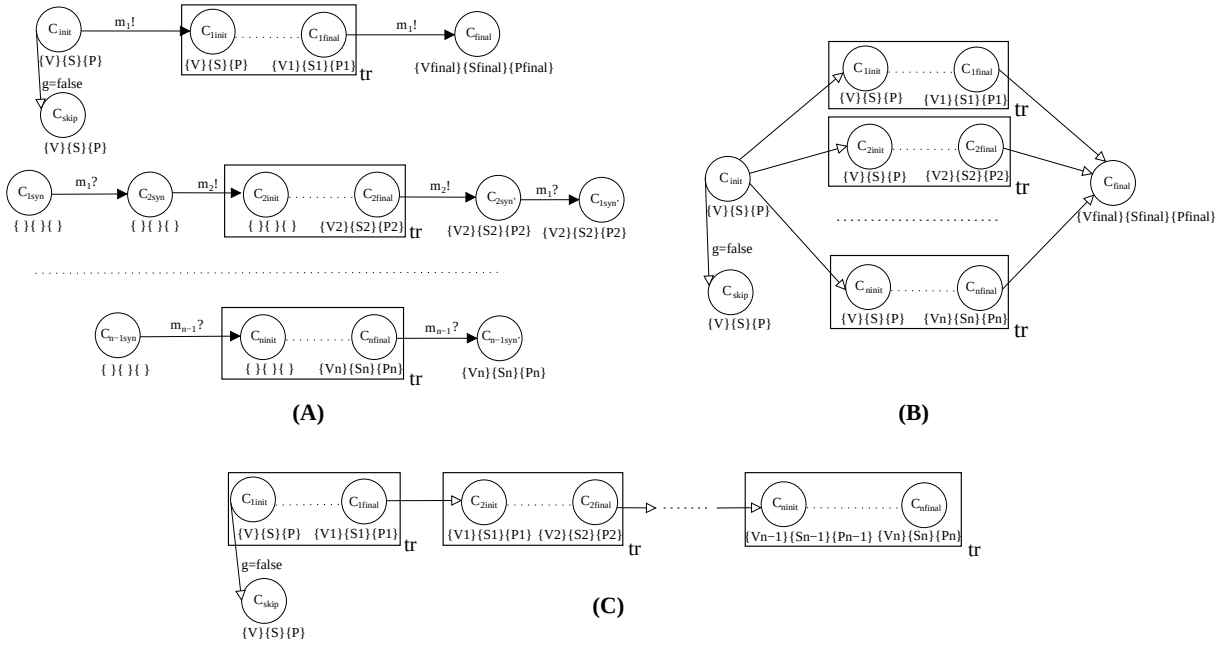

Figure 7: Automata corresponding to the **compositions of deontic norms**

This composition of timed automata is shown graphically in Fig. 7 ***(A)****.*

(9) If several norms are composed by an ***OR-refinement****, that is, we have specified the diagram* $(\varepsilon, name, g, tr, C_1\, Or\, C_2\, Or \ldots Or\, C_n, \varepsilon)$*, their composition corresponds to an automaton in which the automata corresponding to each one of the norms is considered as an alternative. Let us consider* $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ *the automata corresponding to the norms we are composing. The resulting automaton OR∗ preserves the structure of the automata we are composing, adding two nodes called* C_{init} *and* C_{final}*. We define an urgent edge performing no action for each one of the norms we are composing connecting* C_{init} *with the initial node of the automaton corresponding to the norm and we also define an urgent edge performing no action for each one of the norm we are composing connecting the final node of its automaton with* C_{final}*. We add another node* C_{skip} *if guard condition of the parent clause* $g \neq \varepsilon$ *and an urgent edge from* C_{init} *to this new node guarded with the guard condition negated (*$\neg g$*). In the final node of this new structure we keep the violation, satisfaction and permission sets of the previous final node, so we have that Vfinal =* $V1|V2|\ldots|Vn$*, Sfinal =* $S1|S2|\ldots|Sn$ *and Pfinal =* $P1|P2|\ldots|Pn$*. If time restriction of the parent clause* $tr \neq \varepsilon$*, we consider this additional time restriction in all the composed automata together with their own time restrictions. Let* $\mathscr{C}_1 = (N_{\mathscr{C}_1}, n0_{\mathscr{C}_1}, E_{\mathscr{C}_1}, I_{\mathscr{C}_1}), \mathscr{C}_2 = (N_{\mathscr{C}_2}, n0_{\mathscr{C}_2}, E_{\mathscr{C}_2}, I_{\mathscr{C}_2}), \ldots, \mathscr{C}_n = (N_{\mathscr{C}_n}, n0_{\mathscr{C}_n}, E_{\mathscr{C}_n}, I_{\mathscr{C}_n})$*. Considering the case where* $g \neq \varepsilon$ *and* $tr \neq \varepsilon$*, and having that* $E_{\mathscr{C}_1}*, E_{\mathscr{C}_2}*, \ldots, E_{\mathscr{C}_n}*$ *are the sets of edges considering time restriction tr together with their own time restriction, the resulting automaton is therefore* $OR* = (N_{OR*}, n0_{OR*}, E_{OR*}, I_{OR*})$*, where:*

- $N_{OR*} = N_{\mathscr{C}_1} \cup N_{\mathscr{C}_2} \cup \ldots \cup N_{\mathscr{C}_n} \cup \{C_{init}, C_{final}, C_{skip}\}$.
- $n0_{OR*} = C_{1init}$.
- $E_{OR*} = E_{\mathscr{C}_1}* \cup E_{\mathscr{C}_2}* \cup \ldots \cup E_{\mathscr{C}_n}* \cup \{C_{init} \longrightarrow_u C_{1init}, C_{init} \longrightarrow_u C_{2init}, \ldots,$
  $C_{init} \longrightarrow_u C_{ninit}\} \cup \{C_{1final} \longrightarrow_u C_{final}, C_{2final} \longrightarrow_u C_{final}, \ldots,$
  $C_{nfinal} \longrightarrow_u C_{final}\} \cup \{C_{init} \xrightarrow{\neg g}_u C_{skip}\}$.
- $I_{OR*} = I_{\mathscr{C}_1} \cup \{I(n) \equiv x \leq t2+1 \,|\, n \in N_{\mathscr{C}_1} - \{C_{1final}\}\} \cup I_{\mathscr{C}_2} \cup \{I(n) \equiv x \leq t2+1 \,|\, n \in N_{\mathscr{C}_2} - \{C_{2final}\}\} \cup \ldots \cup I_{\mathscr{C}_n} \cup \{I(n) \equiv x \leq t2+1 \,|\, n \in N_{\mathscr{C}_n} - \{C_{nfinal}\}\}$.

This composition of timed automata is shown graphically in Fig. 7 ***(B)****.*

(10) If several norms are composed by a ***SEQ-refinement****, that is, we have specified the diagram* $(\varepsilon, name, g, tr, C_1\, Seq C_2\, Seq \ldots Seq C_n, \varepsilon)$*, their composition corresponds to an automaton in which the automata corresponding to each one of the norms are connected in sequence. Let us consider* $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ *the automata corresponding to the norms we are composing. The resulting automaton* $SEQ*$ *preserves the structure of the automata we are composing, adding just one extra node* C_{skip} *if guard condition of the parent clause* $g \neq \varepsilon$ *and an urgent edge from* C_{1init} *to this new node guarded with the guard condition negated* $(\neg g)$*. We connect with an urgent edge performing no action the ending node of each automaton in the sequence* $(C_{1final}, C_{2final}, \ldots, C_{n-1final})$ *with the initial node of the next automaton* $(C_{2init}, C_{3init} \ldots, C_{ninit})$*. This rule is not applied in the cases of* C_{1init} *(as there is not previous ending node to connect) and* C_{nfinal} *(as there is not following initial node to connect). In the initial node of each one of the composed automata we preserve the violation, satisfaction and permission sets of the previous final node. If time restriction of the parent clause* $tr \neq \varepsilon$*, we consider this additional time restriction in all the composed automata together with their own time restrictions. Let* $\mathscr{C}_1 = (N_{\mathscr{C}_1}, n0_{\mathscr{C}_1}, E_{\mathscr{C}_1}, I_{\mathscr{C}_1}), \mathscr{C}_2 = (N_{\mathscr{C}_2}, n0_{\mathscr{C}_2}, E_{\mathscr{C}_2}, I_{\mathscr{C}_2}), \ldots, \mathscr{C}_n = (N_{\mathscr{C}_n}, n0_{\mathscr{C}_n}, E_{\mathscr{C}_n}, I_{\mathscr{C}_n})$*. Considering the case where* $g \neq \varepsilon$ *and* $tr \neq \varepsilon$*, and having that* $E_{\mathscr{C}_1}*, E_{\mathscr{C}_2}*, \ldots, E_{\mathscr{C}_n}*$ *are the sets of edges considering time restriction* tr *together with their own time restriction, the resulting automaton is* $SEQ* = (N_{SEQ*}, n0_{SEQ*}, E_{SEQ*}, I_{SEQ*})$*, where:*

- $N_{SEQ*} = N_{\mathscr{C}_1} \cup N_{\mathscr{C}_2} \cup \ldots \cup N_{\mathscr{C}_n} \cup \{C_{skip}\}$.
- $n0_{SEQ*} = C_{1init}$.
- $E_{SEQ*} = E_{\mathscr{C}_1}* \cup E_{\mathscr{C}_2}* \cup \ldots \cup E_{\mathscr{C}_n}* \cup \{C_{1init} \xrightarrow{\neg g}_u C_{skip}, C_{1final} \longrightarrow_u C_{2init}, C_{2final} \longrightarrow_u C_{3init}, \ldots, C_{n-1final} \longrightarrow_u C_{ninit}\}$.
- $I_{SEQ*} = I_{\mathscr{C}_1} \cup \{I(n) \equiv x \leq t2+1 \,|\, n \in N_{\mathscr{C}_1} - \{C_{1final}\}\} \cup I_{\mathscr{C}_2} \cup \{I(n) \equiv x \leq t2+1 \,|\, n \in N_{\mathscr{C}_2} - \{C_{2final}\}\} \cup \ldots \cup I_{\mathscr{C}_n} \cup \{I(n) \equiv x \leq t2+1 \,|\, n \in N_{\mathscr{C}_n} - \{C_{nfinal}\}\}$.

This composition of timed automata is shown graphically in Fig. 7 ***(C)****.*

3.1 Implementation in UPPAAL

The implementation of the NTAs we have obtained in UPPAAL is quite straightforward as both, the NTA formalism considered by the tool and the NTA formalism that we have considered, are very similar. There are only a few implementation points that need a more detailed explanation:

- First, as there is no way in UPPAAL of directly expressing that an edge without synchronisation should be taken without delay, that is, there are no urgent edges, we have to find an alternative way of encoding this behaviour. For this purpose we consider the modelling pattern proposed in [3]. The encoding of urgent edges introduces an extra automaton, that we call *Urgent*, with a single location and a self loop. The self loop synchronises on an urgent channel that we call *urg_edge*. An edge can now be made urgent by performing the complimentary action.
- The performance of actions by agents is implemented by means of boolean variables in UPPAAL. We define a boolean variable called *agent_action* for each one of the actions considered in the contract. These variables are initialized to **false** and, when one of the actions is performed by an agent in one of the edges, we update the value of the corresponding variable to **true**.
- Finally, the violation, satisfaction and permission sets are implemented in UPPAAL by means of boolean arrays and constant integers with the names of the clauses of the contract containing obligations, prohibitions or permissions. We define an array *V* for violation, an array *S* for satisfaction,

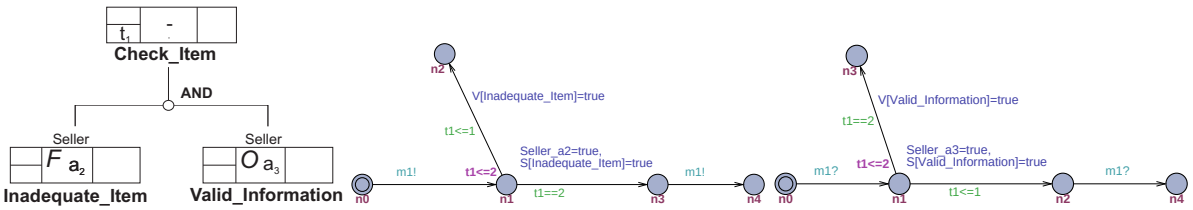

Figure 8: Online auctioning process *C-O Diagram* and corresponding NTA in UPPAAL

and an array P for permission, all of them initialized to **false**. The size of the arrays V and S is equal to the number of obligations and prohibitions in the contract, whereas the size of the array P is equal to the number of permissions. We also define constant integers with the name of the clauses containing obligations and prohibitions, initializing each one of them to a different value (from 0 to the size of the arrays V and S minus 1), and constant integers with the name of the clauses containing permissions, initializing each one of them to a different value (from 0 to the size of the array P minus 1). These constants are used as indexes in the arrays. When taking a transition where the target node contains at least one modified set (an obligation/prohibition is violated, an obligation/prohibition is satisfied or a permission is made effective), we update to **true** in the proper array the value of the index corresponding to the clause. In the case of repairing an obligation/prohibition violation, the index corresponding to the proper clause in V is set to **false**.

3.2 Example: Online Auctioning Process

Let us consider part of a contract about an *online auctioning process*. It specifies that at the beginning of the process the *seller* has **one day** to upload valid information about the item he wants to sell, being forbidden the sale of inadequate items such as replicas of designers items or wild animals. We can identify in this specification an obligation, a prohibition and a real-time constraint affecting both norms. In the representation of this contract as a *C-O Diagram*, that can be seen in the left-hand side of Fig. 8, we have a main clause *Check_Item* including the time restriction **one day**, denoted as t_1. This main clause is decomposed by means of an *AND-refinement*, having on the one hand the clause with the prohibition, called *Inadequate_Item* and denoting the action as $a2$, and on the other hand the clause with the obligation, called *Valid_Information* and denoting the action as $a3$.

By following the *C-O Diagrams* semantics, we can obtain an NTA corresponding to the contract. Its implementation in the UPPAAL tool can be seen in the right-hand side of Fig. 8, having two automata running in parallel, one corresponding to the prohibition and the other one corresponding to the obligation. Now we can take advantage of all the mechanisms for simulation and formal verification provided by the tool to model-check the contract specification. As this is just a small part of a contract, the properties we can verify here are quite obvious. However, this verification process can be very useful over big contracts, verifying properties such as the violation of clauses when a time constraint expires, the possibility of satisfying the contract without violating any clause, etc.

For example, in the current NTA we can check the property that if the *seller* takes more that one day ($t1 > 1$) to upload valid information about the item, the clause *Valid_Information* is always violated. This property is written as follows in the UPPAAL verifier:

$$A1.n1 \quad and \quad t1 > 1 - - > V[Valid\_Information] == true$$

And we obtain that this property is **satisfied**.

4 Conclusions

In this work we have developed a formal semantics for *C-O Diagrams* based on timed automata extended with an ordering of states and edges in order to represent the different deontic modalities. We have also seen how these automata can be implemented in UPPAAL in order to model-check the contract specification, and a small example has been provided.

As future work, we are working on several case studies in order to proove the usefulness of our approach to model-check the specification of complex contracts with real-time constraints. With these case studies we also want to check the complexity of the contracts we can deal with. Finally, we are working on the improvement of the satisfaction rules defined in [7] and their relationship with the *C-O Diagrams* formal semantics.

References

[1] R. Alur & D.L. Dill (1990): *Automata For Modeling Real-Time Systems.* In: *ICALP*, pp. 322–335, doi:10.1007/BFb0032042.

[2] R. Alur & D.L. Dill (1994): *A Theory of Timed Automata*. *Theoretical Computer Science* 126(2), pp. 183–235, doi:10.1016/0304-3975(94)90010-8.

[3] G. Behrmann, A. David & K. G. Larsen (2004): *A tutorial on Uppaal*. *Formal Methods for the Design of Real-Time Systems* (3185), pp. 200–236.

[4] *ebXML: Electronic Business using eXtensible Markup Language*. www.ebxml.org.

[5] J. Hatcliff, G.T. Leavens, k.R.M. Leino, P. Mller & M. Parkinson (2009): *Behavioral Interface Specification Languages*. Technical Report CS-TR-09-01, School of EECS, University of Central Florida.

[6] K. G. Larsen, Z. Pettersson & Y. Wang (1997): *UPPAAL in a Nutshell*. *STTT: International Journal on Software Tools for Technlogy Transfer* 1(1–2), pp. 134–152, doi:10.1007/s100090050010.

[7] E. Martínez, Díaz, G. & Cambronero (2010): *Visual Specification of Formal e-Contracts*. *Fourth Workshop on Formal Languages and Analysis of Contract-Oriented Software (FLACOS 2010)* , pp. 55–61.

[8] E. Martinez, G. Diaz, M. E. Cambronero & G. Schneider (2010): *A Model for Visual Specification of e-Contracts*. In: *The 7th IEEE International Conference on Services Computing (IEEE SCC'10)*, pp. 1–8, doi:10.1109/SCC.2010.32.

[9] P. McNamara (2006): *Deontic Logic*. In: *Gabbay, D.M., Woods, J., eds.: Handbook of the History of Logic*, 7, North-Holland Publishing, pp. 197–289, doi:10.1016/S1874-5857(06)80029-4.

[10] B. Meyer (1986): *Design by Contract*. Technical Report TR-EI-12/CO, Interactive Software Engineering Inc.

[11] J. C. Okika & A. P. Ravn (2008): *Classification of SOA Contract Specification Languages*. In: *2008 IEEE International Conference on Web Services (ICWS'08)*, IEEE Computer Society, pp. 433–440, doi:10.1109/ICWS.2008.36.

[12] C. Prisacariu & G. Schneider (2009): *CL: An Action-based Logic for Reasoning about Contracts*. In: *16th Workshop on Logic, Language, Information and Computation (WOLLIC'09), LNCS* 5514, Springer, pp. 335–349, doi:10.1007/978-3-642-02261-6_27.

[13] *Web Services Agreement Specification (WS-Agreement)*. https://forge.gridforum.org/projects/graap-wg/document/WS-AgreementSpecification/en/7.

[14] *WSLA: Web Service Level Agreements*. www.research.ibm.com/wsla/.