Themulus: A Timed Contract-Calculus[a]

Alberto Aranda García[1], María-Emilia Cambronero[2], Christian Colombo[1], Luis Llana[3], and Gordon J. Pace[1]

[1]*Department of Computer Science, University of Malta, Malta*

[2]*Department of Computer Science, University of Castilla-La Mancha, Albacete, Spain*

[3]*Department of Computer Science and the Knowledge Technology Institute, University Complutense of Madrid, Madrid, Spain*

alb.aranda@gmail.com, MEmilia.Cambronero@uclm.es, Christian.Colombo@um.edu.mt, llana@ucm.es, Gordon.Pace@um.edu.mt

Keywords: Deontic Logic, Formal Methods, Operational Semantics, Simulation Semantics, Themulus

Abstract: Over these past years, formal reasoning about contracts between parties has been increasingly explored in the literature. There has been a shift of view from that viewing contracts simply as properties to be satisfied by the parties to contracts as first class syntactic objects which can be reasoned about independently of the parties' behaviour. In this paper, we present a real-time deontic contract calculus, Themulus, to reason about contracts, abstracting the parties' behaviour through the use of a simulation relation. In doing so, we can compare real-time deontic contracts in terms of their strictness over permissions, prohibitions and obligations.

1 Introduction

The need for formal techniques for reasoning about contracts is becoming increasingly important as software systems interact more frequently with other systems and with our everyday life. Although for many applications a property-based approach suffices — specifying pre-/post-conditions, invariants, temporal properties, etc. — other applications require a first class notion of contracts which property-based approaches do not address sufficiently well. Deontic logics [Georg Henrik Von Wright, 1951] have been developed precisely to deal with such a need to talk about *ideal* behaviour of a system, possibly also including exceptional situations when the system deviates from such behaviour. For instance, consider a contract which specifies that a party is to perform a particular action, but if they fail to do so, they will incur an additional charge (which they are obliged to pay) and prohibited from taking certain actions until they do so. Such contracts, typically using a deontic logic, have been referred to as *total contracts* and have been argued to be more informative (with the right abstractions) than simple properties [Fenech et al., 2009]. The opportunity to move from seeing specifications simply as expressions in a logic which must hold to the higher level view of them being a form of contract goes back to Khosla [Khosla, 1988]. By looking at contracts as first-class entities which can be reasoned about, manipulated, etc., one can perform contract analysis independent of the systems the contract will regulate, e.g., one can analyze contracts for potential conflicts, or to evaluate which is the stricter one.

Different approaches to contract analysis have been reported in the literature, with most approaches focusing on the violation semantics of contracts, thus enabling the characterization of agreements between parties or agents regulating their behaviour. In systems with interacting parties, contracts play an even more important role since an agent's behaviour (or non-behaviour) directly impacts other agents. Surprisingly, many contract logics reason about deontic modalities such as obligations and permissions without specifying agents, and the literature addressing reasoning about directed deontic modalities is relatively sparse (e.g. Modal Action Logic [Jeremaes et al., 1986], deontic STIT logic [Belnap and Perloff, 1993, Horty, 2001], Business Contract Language [Governatori and Milosevic, 2005], contract

[a]This work has been supported by the Spanish MINECO-FEDER (grant numbers DArDOS, TIN2015-65845-C3-1-R and FAME, RTI2018-093608-B-C31) and the Region of Madrid (grant number FORTE-CM, S2018/TCS-4314)

automata [Pace and Schapachnik, 2012]).

Interaction has long been studied in computer science using calculi to reason about communicating transition systems enabling the classification of systems into correct and incorrect ones with respect to a property. It is only recently, however, that the distinction between properties and contracts has started being explored. Yet, in much of the literature, contract comparison is still defined in terms of how the contracts regulate systems e.g. saying that a contract is stricter than another if any system which violates the latter will also violate the former. This means that to reason about contracts one has to bring to play the systems which they regulate.

Orthogonal to this issue is that of the notion of time in contracts. From work in linear temporal logics, one can (broadly) categorize such logics into a number of categories: (i) ones which permit reasoning about sequentiality of events; (ii) ones which can also reason about time using a notion of a discrete global clock; and (iii) ones which allow reasoning about timers which can take continuous time values and which can interact with such timers e.g. triggering on timeouts, or resetting the timers. The notion of continuous time clocks, i.e. (iii), introduces additional complexity including aspects which may be undecidable as can be seen, for instance, in the extensive work on verification of timed automata and hybrid systems in general [Asarin et al., 2012]. Multi-party session types, which share much with contracts have been extended to deal with timed aspects [Bocchi et al., 2014]. Our approach to time shares much with theirs, although our handling of notions such as permission allows for an implicit notion of deontic modalities

Furthermore, if events are timed, one has to introduce a notion of time in the deontic logic — whether in a point-wise manner (e.g. an obligation to perform a particular action at a particular time) or over time intervals (e.g. an obligation to perform a particular action before a deadline). There is much work about the combination of discrete time temporal and deontic logics, but less so with dense-time logics. Our approach is an interval logic one, taking the approach adopted by real time logics such as duration calculus [Chaochen et al., 1991], which only allows statements about signal values over non-point intervals.

In earlier work, we have developed a calculus to reason about contracts independently of the systems [Cambronero et al., 2017] in which, only temporal sequentiality of events was handled. In this paper, we present a time extension, give an operational view of contracts, and use simulation techniques to reason about contracts at an operational level.

The paper is organised as follows. First, we present a running example (section 2) used throughout the paper to clarify concepts. Then, the notation we will use to formalize our notions is presented in Section 3. We then present our timed contract calculus Themulus in Section 4 and formalize the notion of refinement of contracts in section 5. We finally conclude in Section 6 with some conclusions and possible lines of future work.

2 Running Example

In the rest of the paper, we will illustrate our logic and results based on a contract commonly used in the literature, that of a plane boarding system, based on e.g. [Azzopardi et al., 2014]. In this section we present this use case — an agreement between the passenger and airline company, regulating the plane boarding process, from check-in till the flight, including time constraints. The use case is a simplified version based on the Madrid Barajas airport regulations.

1. The passenger is permitted to use the check-in desk within two hours before the plane takes off (t_0).
2. At the check-in desk, the passenger is obliged to present her boarding pass whitin 5 minutes.
3. After presenting the boarding pass, the passenger must show her passport, she has 5 minutes for this purpose.
4. Henceforth, the passenger is (i) prohibited from carrying liquids in her hand-luggage until boarding; and (ii) prohibited from carrying weapons during the whole trip until the plane lands. If she has liquids in her hand-luggage, she is obliged to dispose of them within 10 minutes.
5. After presenting her passport, the passenger is permitted to board within 90 minutes and to present the hand-luggage to the staff within 10 minutes. Therefore, the airline company is obliged to allow the passenger to board within 90 minutes. If the passenger is stopped from carrying luggage, the airline company is obliged to put the passenger's hand luggage in the hold within 20 minutes.

3 Background and Notation

Contracts regulate the behaviour of agents or parties that are acting concurrently. In this Section, we present notation used to describe these agents and their behaviour in order to be able to formalize contracts in the following sections.

Structurally, the underlying system consists of several indexed agents running in parallel, using variables A, A' to represent the individual agents. The system as a whole will consist of the parallel composition

of all agents indexed by a finite set I i.e. the system will be of the form $||_{i\in I}A_i$. We will use variables $\mathcal{A}$, $\mathcal{A}'$ to denote the state of the system as a whole.

Notation: The visible behaviour of the system and agents will be assumed to consist of actions over Act, and the agents' behaviour will be assumed to consist of (i) a relation indicating how their state changes whenever such action occurs; and (ii) a relation indicating how they change over time. Time will be taken to range over the non-negative reals: $\mathbb{T} = \mathbb{R}^+$. Agents semantics are thus represented as *timed labelled transition systems*:

- $A \xrightarrow{a} A'$, for $a \in \mathsf{Act}$, indicates that agent A changes to A' upon performing action a. As it is usual in process algebrae [Yi, 1991], the execution of actions does not consume time. The transition $A \xrightarrow{a}\!\!\!\!\!\rightarrow$ indicates that agent A cannot perform a: $A \xrightarrow{a}\!\!\!\!\!\rightarrow \stackrel{\text{df}}{=} \neg\exists A' \cdot A \xrightarrow{a} A'$.
- $A \overset{d}{\rightsquigarrow} A'$, for $d > 0 \in \mathbb{T}$, indicates that agent A evolves to A' after d time units pass.

Assumptions: We will assume that agents are non-blocking: for any agent A, there is an agent state A' such that either (i) $A \xrightarrow{a} A'$ (for some $a \in \mathsf{Act}$); or (ii) $A \overset{d}{\rightsquigarrow} A'$ (for some $d > 0 \in \mathbb{T}$). We also assume the following common properties of the time transition relation: time determinism, time additivity, and time continuity.

We can now define how a system as a whole (a composition of agents) evolves. There are two kinds of transitions: (i) action transitions of the form $\mathcal{A} \xrightarrow{a,S} \mathcal{A}'$ will indicate that system $\mathcal{A}$ can perform action $a \in \mathsf{Act}$ with agents indexed by $S \in 2^I$ participating, to become system $\mathcal{A}'$; and (ii) timed transitions $\mathcal{A} \overset{d}{\rightsquigarrow} \mathcal{A}'$ indicate the evolution of the system as a whole over time.

Definition 1 *We define the following transition relations over systems:*

- $\mathcal{A} \xrightarrow{a,S} \mathcal{A}'$, *with* $S \in 2^I$ *indicating that agents in* S *(and no others) synchronise on action* a. *Formally,* $\mathcal{A} \xrightarrow{a,S} \mathcal{A}'$, *where* $\mathcal{A} = A_1 \parallel A_2 \parallel \cdots \parallel A_n$, *and* $\mathcal{A}' = A'_1 \parallel A'_2 \cdots \parallel A'_{n'}$ *is defined as follows: (i) the number of agents does not change:* $n = n'$; *(ii) agents other than those whose index appears in* S *do not participate in action* a: $\forall i \in I \cdot i \notin S \Rightarrow A_i = A'_i$; *and (iii) agents indexed in* S *evolve over action* a: $\forall i \in I \cdot i \in S \Rightarrow A_i \xrightarrow{a} A'_i$.
- $\mathcal{A} \overset{d}{\rightsquigarrow} \mathcal{A}'$ *indicates that system* $\mathcal{A}$ *evolves to* $\mathcal{A}'$ *after* $d > 0 \in \mathbb{T}$ *time units pass. Formally we define* $\mathcal{A} \overset{d}{\rightsquigarrow} \mathcal{A}'$ *to mean that all agents evolve with a time transition of length* d: $A_i \overset{d}{\rightsquigarrow} A'_i$ *for all* $i \in I$, *where* $\mathcal{A} = A_1 \parallel \cdots \parallel A_l \parallel \cdots \parallel A_n$, *and* $\mathcal{A}' = A'_1 \parallel \cdots \parallel A'_l \parallel \cdots \parallel A'_n$.

We will also write $\mathcal{A} \xrightarrow{a,S}$ to mean that system $\mathcal{A}$ can perform action a involving the agents in set S: $\mathcal{A} \xrightarrow{a,S} \stackrel{\text{df}}{=} \exists \mathcal{A}' \cdot \mathcal{A} \xrightarrow{a,S} \mathcal{A}'$. The lack of such a transition is written as: $\mathcal{A} \xrightarrow{a,S}\!\!\!\!\!\rightarrow$.

In order to formalize violation of contracts, we will use predicates over agent behaviour.

Definition 2 *A predicate is defined in terms of the following grammar:*

$$\mathcal{P} ::= tt \mid ff \mid \langle a,k\rangle \mid \langle a,\overline{k}\rangle \mid P \vee Q \mid P \wedge Q$$

In the grammar above, $k \in I$ *ranges over agent indices,* $a \in \mathsf{Act}$ *over actions, and* $P, Q \in \mathcal{P}$ *over predicates.*

Predicates *tt* and *ff* denote true and false respectively. Predicate $\langle a,k\rangle$ means that agent k may perform action a. However, since some actions may require involvement by several agents, we use the predicate $\langle a,\overline{k}\rangle$ to indicate that agent k wants to perform action a, but this action is not offered by any other agent for synchronisation. For instance, an agent c may want to purchase a ticket (action: *ticket*) to go to a theatre. Predicate $\langle ticket, c\rangle$ indicates the success of such an action with c participating. However, if the action requires the participation of the ticket office, we can write the predicate $\langle ticket, \overline{c}\rangle$ to indicate that c wanted to perform the action, but neither the ticket office (nor any other agent) was willing to perform the handshake required. Predicate disjunction and conjunction are indicated by $P \vee Q$ and $P \wedge Q$ respectively.

Definition 3 *The semantics of a predicate* P *under a system* $\mathcal{A}$, *written* $\mathcal{A} \models P$, *is defined as follows:*

$$\begin{array}{lcl}
\mathcal{A} \models tt & \stackrel{df}{=} & true \\
\mathcal{A} \models ff & \stackrel{df}{=} & false \\
\mathcal{A} \models \langle a,k\rangle & \stackrel{df}{=} & \exists S \in 2^I,\ \mathcal{A}' \cdot \mathcal{A} \xrightarrow{a,S} \mathcal{A}' \wedge k \in S \\
\mathcal{A} \models \langle a,\overline{k}\rangle & \stackrel{df}{=} & \mathcal{A} \xrightarrow{a,\{k\}} \text{ and } \forall l \cdot l \neq k \Rightarrow A_l \xrightarrow{a}\!\!\!\!\!\rightarrow \\
\mathcal{A} \models P \vee Q & \stackrel{df}{=} & \mathcal{A} \models P \text{ or } \mathcal{A} \models Q \\
\mathcal{A} \models P \wedge Q & \stackrel{df}{=} & \mathcal{A} \models P \text{ and } \mathcal{A} \models Q
\end{array}$$

We can now define the notion of *stronger-than* and that of *equivalence* between predicates.

Definition 4 *Given predicates* $P, Q \in \mathcal{P}$, *we say that* P *is* stronger *than* Q, *written* $P \models Q$, *iff for any state of any system* $\mathcal{A}$ *for which* $\mathcal{A} \models P$ *holds,* $\mathcal{A} \models Q$ *also holds. We say that* P *is* equivalent *to* Q, *written* $P \dashv\models Q$, *iff* $P \models Q$ *and* $Q \models P$.

We will now present a proposition which indicates how equivalence combines with disjunction and action success.

Proposition 1 *Let $P,Q \in \mathcal{P}$, k be an agent index and $a \in \mathsf{Act}$, then*

1. *If $P \not\rightleftharpoons tt$, then $\langle a,k\rangle \vee P \not\rightleftharpoons tt$ and $\langle a,\overline{k}\rangle \vee P \not\rightleftharpoons tt$.*
2. *If $P \vee Q \dashv\models tt$ then $P \dashv\models tt$ or $Q \dashv\models tt$.*

4 A Timed Contract Calculus

We can now define our contract calculus Themulus. We start by defining its syntax and an equivalence relation over the syntactic forms. We then define the notion of contract violation conditions based on the operational semantics of the calculus. As we mentioned before, we will assume a time domain $\mathbb{T}$ ranging over the non-negative reals. In order to deal with the recursion operator, we assume a set of variables fvars over which recursion will be defined.

4.1 Contract Syntax

Definition 5 *The set of contract formulae denoted by C (with variable $\varphi \in C$ to range over the contracts) is syntactically defined as follows:*

$$\begin{array}{rcl} \varphi & ::= & \top \mid \bot \mid \mathcal{P}_k(a)[d] \mid O_k(a)[d] \mid \mathcal{F}_k(a)[d] \\ & \mid & \mathsf{wait}(d) \mid \mathit{cond}_k(a)[d](\varphi_1,\varphi_2) \mid \varphi_1;\varphi_2 \\ & \mid & \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \blacktriangleright \varphi_2 \mid \mathsf{rec}\, x.\varphi \mid x \end{array}$$

where $a \in \mathsf{Act}$, $x \in \mathsf{fvars}$, $k \in I$ and $d \in \mathbb{T} \cup \{\infty\}$.

The basic formulae $\top$ and $\bot$ indicate, respectively, the contracts that are trivially satisfied and violated. The key modalities we use from deontic logic to specify contracts are permissions, obligations, and prohibitions. The formula $\mathcal{P}_k(a)[d]$ indicates the permission of agent k to perform action a within d time units, while $O_k(a)[d]$ is an obligation on agent k to perform action a within d time units, and $\mathcal{F}_k(a)[d]$ is the prohibition on agent k to perform action a within d time units. The formula $\mathsf{wait}(d)$ represents a delay of d time units.

Contract disjunction is written as $\varphi_1 \vee \varphi_2$, and contract conjunction as $\varphi_1 \wedge \varphi_2$. The formula $\varphi_1;\varphi_2$ indicates the sequential composition of two contracts — in order to satisfy the whole contract, the first contract φ_1 must be satisfied and then the second one φ_2. For instance, we can model the obligation of agent k of doing action a in 3 time units after a delay of 2 time units: $\mathsf{wait}(2);O_k(a)[3]$.

The reparation operator, written $\varphi_1 \blacktriangleright \varphi_2$, is the contract which starts off as φ_1, but when violated triggers contract φ_2, e.g., $O_1(a)[2] \blacktriangleright \mathcal{P}_2(b)[5]$ is the contract which obliges agent 1 to perform action a in 2 time units, but if she does not, permits agent 2 to perform action b in 5 time units.

The formula $\mathrm{cond}_k(a)[d](\varphi_1,\varphi_2)$ is a conditional contract where (i) if party k performs action a within d time units it proceeds to behave like φ_1; otherwise (ii) if d time units elapse without a being performed by k, it then proceeds to behave like φ_2. Note that we can generalize to more general conditions on the system, but we limit it to the ability of a party to perform an action for the scope of this paper.

Finally, $\mathsf{rec}\, x.\varphi$ and x handles recursive contracts, e.g., $\mathsf{rec}\, x.O_p(a)[d];x$ is the contract which obliges agent p to repeatedly perform action a within d time units of each other. In contrast, $\mathsf{rec}\, x.O_r(pr)[10] \wedge \mathsf{wait}(30);x$, is the contract in which agent r is repeatedly obliged to pay rent (action pr) during the first 10 days of the month.

Using these basic contract combinators, we can define more complex ones, for example, a prohibition which persists until a particular action is performed — a prohibition on agent k from performing action a until party l performs action b, written $\mathcal{F}([a,k]\,\mathcal{U}[b,l])$, and defined as follows:

$$\mathcal{F}([a,k]\,\mathcal{U}[b,l]) \stackrel{\text{df}}{=} \mathsf{rec}\, x.\big(\mathrm{cond}_k(a)[\infty](\bot,\top) \wedge \mathrm{cond}_l(b)[\infty](\top,x)\big)$$

Example 1 The contract of the plane boarding system from Section 2, can be formalised using our contract calculus as follows:

$$\begin{array}{rcl} \varphi_0 & ::= & \mathcal{P}_p(\mathtt{checkin})[t_0 - 120] \\ \varphi_1 & ::= & O_p(\mathtt{PBP})[5] \\ \varphi_2 & ::= & O_p(\mathtt{ShP})[5] \\ \varphi_3 & ::= & (\mathcal{F}([\mathtt{weapon},p]\,\mathcal{U}[\mathtt{landing},c])) \wedge \\ & & ((\mathcal{F}([\mathtt{lq},p]\,\mathcal{U}[\mathtt{boarding},p])) \blacktriangleright O_p(\mathtt{dlq})[10]) \\ \varphi_4 & ::= & (\mathcal{P}_p(\mathtt{brd})[90];\mathcal{P}_p(\mathtt{hl})[10]) \blacktriangleright \\ & & (O_c(\mathtt{brd})[90];\, O_c(\mathtt{hlhld})[20]) \\ \\ \mathit{PBS} & ::= & \varphi_0;\varphi_1;\varphi_2;(\varphi_3 \wedge \varphi_4) \end{array}$$

Where t_0 is departure estimated time. Note that the clauses φ_0 to φ_4 are used to express different parts of the contract, and combined together in the top-level contract expression *PBS*. □

The syntax of our logic allows for formulae whose meaning is unclear. For instance, the formula $\mathcal{F}_k(a)[d] \vee x$ is not well-formed since it contains a free instance of variable x. Another problem arises with formulae such as $\mathsf{rec}\, x.\mathcal{F}_k(a)[d] \vee x$, which use recursion not guarded by a prefix formula since the latter ensures certain desirable properties of our operational

semantics. In order to simplify our semantics, we restrict the set of well-formed formulae to ones which are (i) *closed*; and (ii) *strongly prefixed*. As usual, the closed formulae are those that do not contain free recursion variables (a recursion variable x is free if it not bound to a rec x above it). A strongly prefixed formula is one where all the occurrences of the formula variables are prefixed by an obligation, prohibition, permission or wait operation.

4.2 Syntactical Congruence

As in other such approaches [Milner, 1999], we start by defining a syntactical congruence, denoted by $\equiv$, between contracts. This congruence is to be applied on a well-formed formula and its subformulae before the rules of the operational semantics.

Definition 6 *We define the relation $\equiv \subseteq \mathcal{C} \times \mathcal{C}$ as the least congruence relation that includes:*

1. $\varphi \wedge \top \equiv \varphi$
2. $\top \wedge \varphi \equiv \varphi$
3. $\bot \wedge \varphi \equiv \bot$
4. $\varphi \wedge \bot \equiv \bot$
5. $\varphi \vee \top \equiv \top$
6. $\top \vee \varphi \equiv \top$
7. $\varphi \vee \bot \equiv \varphi$
8. $\bot \vee \varphi \equiv \varphi$
9. $\top ; \varphi \equiv \varphi$
10. $\bot ; \varphi \equiv \bot$
11. $\top \triangleright \varphi \equiv \top$
12. $\bot \triangleright \varphi \equiv \varphi$
13. $O_k(a)[0] \equiv \bot$
14. $\mathcal{F}_k(a)[0] \equiv \top$
15. $\mathcal{P}_k(a)[0] \equiv \top$
16. $\mathsf{wait}(0) \equiv \top$
17. $cond_k(a)[0](\varphi, \psi) \equiv \psi$

In order to compute the $\equiv$ relation, we transform it into a rewriting calculus: we can see the rules above as rewriting rules going from left to right. For instance, the equivalence rule 13 ($O_k(a)[0] \equiv \bot$) allows us to rewrite $O_k(a)[0]; \mathcal{P}_l(b)[5]$ to $\bot; \mathcal{P}_l(b)[5]$, which in turn can be rewritten to $\bot$ using rule 10 ($\bot ; \varphi \equiv \bot$).

Definition 7 *We write $\varphi \hookrightarrow \varphi'$ (where $\varphi, \varphi' \in \mathcal{C}$), if φ' is the result of applying one of the equivalence rules from left to right on a subexpression of φ.*

Example 2 Returning to the plane boarding system agreement, consider the obligation on passengers to present the boarding pass (action PBP) within 5 time units: $O_p(\mathsf{PBP})[5]$. In this case, equivalence rule 13 can be applied after 5 time units: $O_p(\mathsf{PBP})[5] \overset{5}{\rightsquigarrow} O_p(\mathsf{PBP})[0] \hookrightarrow \bot$. □

In order to justify the simplification of contract formulae by applying these rules repeatedly, we will need to prove that the rewriting process is terminating and confluent. To prove confluence of $\hookrightarrow$, we will first prove *local confluence*, from which confluence follows using a standard result from computer science.

Proposition 2 *The $\hookrightarrow \in \mathcal{C} \leftrightarrow \mathcal{C}$ relation is: (i)* terminating*: there is no infinite sequence $\varphi_1, \varphi_2 \ldots$, such that $\forall i \cdot \varphi_i \hookrightarrow \varphi_{i+1}$; and (ii)* locally confluent*: if $\varphi \hookrightarrow \varphi_1$ and $\varphi \hookrightarrow \varphi_2$, then there exists a contract φ' such that $\varphi_1 \hookrightarrow^* \varphi'$ and $\varphi_2 \hookrightarrow^* \varphi'$.*

Proof. Since the right term is always syntactically smaller than the one on the left, the relation $\hookrightarrow$ is a well-founded relation, and thus, *termination* is easily proved. Local confluence is proved by structural induction on φ. The base cases are trivial. To prove the inductive cases, we perform case by case analysis on the different rules, which are applied to the subformulae to show that the confluence result holds. □

Based on these results, confluence of $\hookrightarrow$ follows using Newman's Lemma [Newman, 1942].

Corollary 1 *The syntactic equivalence relation applied from left to right is confluent: if $\varphi \hookrightarrow^* \varphi_1$ and $\varphi \hookrightarrow^* \varphi_2$, then there is a contract φ' such that $\varphi_1 \hookrightarrow^* \varphi'$ and $\varphi_2 \hookrightarrow^* \varphi'$.*

Confluence and termination mean that any given formula can be deterministically reduced to an *irreducible* formula in a finite number of steps.

Definition 8 *A contract formula $\varphi \in \mathcal{C}$ is said to be* irreducible*, if the equivalence relation cannot be applied to any of its subexpressions: $\neg \exists \varphi' \in \mathcal{C} \cdot \varphi \hookrightarrow \varphi'$.*

Given contract formulae $\varphi, \varphi' \in \mathcal{C}$, we write $\varphi \longmapsto \varphi'$ iff (i) φ can be syntactically reduced to φ' in a number of steps: $\varphi \hookrightarrow^* \varphi'$; and (ii) φ' is irreducible.

Confluence and termination guarantee that for a given φ, there exists a unique φ' such that $\varphi \longmapsto \varphi'$.

4.3 Operational Semantics

We can now define an operational semantics for our contract calculus. The rules of the operational semantics appear in Figure 1. The semantics take one of three forms: (i) $\varphi \xrightarrow{a,k} \varphi'$ to denote that contract φ can evolve (in one step) to φ' when action a is performed, which involves party k (and possibly other parties); or (ii) $\varphi \xrightarrow{\overline{(a,k)}} \varphi'$ indicating that the contract φ can evolve to φ' when the action a is not offered by any party other than k; or (iii) $\varphi \overset{d}{\rightsquigarrow} \varphi'$ to represent that contract φ can evolve to contract φ' when d time units pass. We will use variable α to stand for a label of either form: (a,k) or $\overline{(a,k)}$. The rules of the operational semantics are always applied to irreducible terms.

The core of any contract reasoning formalism is the rules defining the semantics of the deontic modalities.

Rules (**O1**), (**O2**), (**O3**), (**O4**), and (**O5**) define the behaviour of obligations $O_k(a)[d]$, i.e., the obligation on agent k to perform action a within d time units.

(O1)	$O_k(a)[d] \xrightarrow{a,k} \top$
(O2)	$O_k(a)[d] \xrightarrow{\overline{(a,k)}} \top$
(O3)	$O_k(a)[d] \xrightarrow{b,l} O_k(a)[d], (a,k) \neq (b,l)$
(O4)	$O_k(a)[d] \xrightarrow{\overline{(b,l)}} O_k(a)[d], (a,k) \neq (b,l)$
(O5)	$O_k(a)[d] \overset{d'}{\rightsquigarrow} O_k(a)[d-d'], 0 < d' \leqslant d$

(P1)	$\mathcal{P}_k(a)[d] \xrightarrow{a,k} \top$
(P2)	$\mathcal{P}_k(a)[d] \xrightarrow{b,l} \mathcal{P}_k(a)[d], \quad (a,k) \neq (b,l)$
(P3)	$\mathcal{P}_k(a)[d] \xrightarrow{\overline{(a,k)}} \bot$
(P4)	$\mathcal{P}_k(a)[d] \xrightarrow{\overline{(b,l)}} \mathcal{P}_k(a)[d], \quad (a,k) \neq (b,l)$
(P5)	$\mathcal{P}_k(a)[d] \overset{d'}{\rightsquigarrow} \mathcal{P}_k(a)[d-d'], 0 < d' \leqslant d$

(AO1)	$\dfrac{\varphi \xrightarrow{\alpha} \varphi',\ \psi \xrightarrow{\alpha} \psi'}{\varphi \operatorname{op} \psi \xrightarrow{\alpha} \varphi' \operatorname{op} \psi'}$
(AO2)	$\dfrac{\varphi \overset{d}{\rightsquigarrow} \varphi',\ \psi \overset{d}{\rightsquigarrow} \psi'}{\varphi \operatorname{op} \psi \overset{d}{\rightsquigarrow} \varphi' \operatorname{op} \psi'}$
(AO3)	$\dfrac{\varphi \overset{d}{\rightsquigarrow} \top,\ \psi \overset{d'}{\rightsquigarrow} \psi'}{\varphi \wedge \psi \overset{d'}{\rightsquigarrow} \psi'},\ d' \geqslant d$
(AO4)	$\dfrac{\varphi \overset{d}{\rightsquigarrow} \varphi',\ \psi \overset{d'}{\rightsquigarrow} \top}{\varphi \wedge \psi \overset{d}{\rightsquigarrow} \varphi'},\ d \geqslant d'$
(AO5)	$\dfrac{\varphi \overset{d}{\rightsquigarrow} \bot,\ \psi \overset{d'}{\rightsquigarrow} \psi'}{\varphi \vee \psi \overset{d'}{\rightsquigarrow} \psi'},\ d' \geqslant d$
(AO6)	$\dfrac{\varphi \overset{d}{\rightsquigarrow} \varphi',\ \psi \overset{d'}{\rightsquigarrow} \bot}{\varphi \vee \psi \overset{d}{\rightsquigarrow} \varphi'},\ d \geqslant d'$

(wait1)	$\mathsf{wait}(d) \overset{d'}{\rightsquigarrow} \mathsf{wait}(d-d'), 0 < d' \leqslant d$
(wait2)	$\mathsf{wait}(d) \xrightarrow{\alpha} \mathsf{wait}(d)$

(F1)	$\mathcal{F}_k(a)[d] \xrightarrow{a,k} \bot$
(F2)	$\mathcal{F}_k(a)[d] \xrightarrow{\overline{(a,k)}} \bot$
(F3)	$\mathcal{F}_k(a)[d] \xrightarrow{b,l} \mathcal{F}_k(a)[d], \quad (b,l) \neq (a,k)$
(F4)	$\mathcal{F}_k(a)[d] \xrightarrow{\overline{(b,l)}} \mathcal{F}_k(a)[d], \quad (b,l) \neq (a,k)$
(F5)	$\mathcal{F}_k(a)[d] \overset{d'}{\rightsquigarrow} \mathcal{F}_k(a)[d-d'], 0 < d' \leqslant d$

(C1)	$\mathrm{cond}_k(a)[d](\varphi,\psi) \xrightarrow{a,k} \varphi$
(C2)	$\mathrm{cond}_k(a)[d](\varphi,\psi) \xrightarrow{\overline{(a,k)}} \varphi$
(C3)	$\mathrm{cond}_k(a)[d](\varphi,\psi) \xrightarrow{b,l} \psi, \quad (b,l) \neq (a,k)$
(C4)	$\mathrm{cond}_k(a)[d](\varphi,\psi) \xrightarrow{\overline{(b,l)}} \psi, \quad (b,l) \neq (a,k)$
(C5)	$\mathrm{cond}_k(a)[d](\varphi,\psi) \overset{d'}{\rightsquigarrow} \mathrm{cond}_k(a)[d-d'](\varphi,\psi), \quad 0 < d' \leqslant d$

(V1)	$\dfrac{\varphi \xrightarrow{\alpha} \varphi'}{\varphi \triangleright \psi \xrightarrow{\alpha} \varphi' \triangleright \psi}$
(V2)	$\dfrac{\varphi \overset{d}{\rightsquigarrow} \varphi'}{\varphi \triangleright \psi \overset{d}{\rightsquigarrow} \varphi' \triangleright \psi}$
(V3)	$\dfrac{\varphi \overset{d}{\rightsquigarrow} \bot, \psi \overset{d'}{\rightsquigarrow} \psi'}{\varphi \triangleright \psi \overset{d+d'}{\rightsquigarrow} \psi'}$
(S1)	$\dfrac{\varphi \xrightarrow{\alpha} \varphi'}{\varphi;\psi \xrightarrow{\alpha} \varphi';\psi}$
(S2)	$\dfrac{\varphi \overset{d}{\rightsquigarrow} \varphi'}{\varphi;\psi \overset{d}{\rightsquigarrow} \varphi';\psi}$
(S3)	$\dfrac{\varphi \overset{d}{\rightsquigarrow} \top,\ \psi \overset{d'}{\rightsquigarrow} \psi'}{\varphi;\psi \overset{d+d'}{\rightsquigarrow} \psi'}$

(REC1)	$\dfrac{\varphi \xrightarrow{\alpha} \varphi'}{\mathsf{rec}\, x.\varphi \xrightarrow{\alpha} \varphi'[x/\mathsf{rec}\, x.\varphi]}$
(REC2)	$\dfrac{\varphi \overset{d}{\rightsquigarrow} \varphi'}{\mathsf{rec}\, x.\varphi \overset{d}{\rightsquigarrow} \varphi'[x/\mathsf{rec}\, x.\varphi]}$

Figure 1: Operational Semantics transition rules

Rules (**O1**) and (**O2**) handle the case of the obligation clause being satisfied when agent k does action a within d time units, in this case, the contract reduces to the trivially satisfied one ($\top$). Rules (**O3**) and (**O4**) consider the case when another agent l performs an action ($l \neq k$) or the action b is not the compulsory one $b \neq a$; in both cases the obligation remaining intact. Let us recall that actions are instantaneous, so the time constraints do not change. Finally, (**O5**) handles the case when d' time units pass with $d' \leqslant d$, then the obligation remains, but the obligation time decreases in d' time units. Recall that $O_k(a)[0]$ is handled through syntactic equivalence ($\equiv \bot$).

Example 3 Let us consider the obligation on the passenger (agent: p) to present the boarding pass (action PBP) within 5 time units: $O_p(\mathtt{PBP})[5]$. The possible outcomes are: (i) rule (**O1**) applies if the passenger presents the boarding pass within 5 time units, with the contract evolving to $\top$: $O_p(\mathtt{PBP})[5] \xrightarrow{\mathtt{PBP},p} \top$; (ii) rule (**O2**) can be applied if the passenger is not allowed to perform the action: $O_p(\mathtt{PBP})[5] \xrightarrow{\overline{(\mathtt{PBP},p)}} \top$; (iii) if an action other than PBP is performed or PBP is performed by another party, the obligation remains intact by rule (**O3**): $O_p(\mathtt{PBP})[5] \xrightarrow{b,l} O_p(\mathtt{PBP})[5]$ (where $b \neq \mathtt{PBP}$ or $l \neq p$); (iv) similarly if other parties or actions are not allowed, the obligation remains unchanged by rule (**O4**): $O_p(\mathtt{PBP})[5] \xrightarrow{\overline{(b,l)}} O_p(\mathtt{PBP})[5]$ (where $b \neq \mathtt{PBP}$ or $l \neq p$); and finally (v) rule (**O5**) handles when an amount of time less than 5 time units elapses, in which case the obligation remains in force, but the deadline is moved accordingly: $O_p(\mathtt{PBP})[5] \overset{\delta}{\rightsquigarrow} O_p(\mathtt{PBP})[5-\delta]$ (where $\delta \leqslant 5$). Note that in this final case, when the deadline of the obligation decreases to 0, the syntactic equivalence $O_p(\mathtt{PBP})[0] \equiv \bot$ is directly applied and reduced accordingly.

□

Rules (**F1**), (**F2**), (**F3**), (**F4**), and (**F5**) define the cases for prohibition similar to obligation.

Permission of agent k to perform action a within

d time units ($\mathcal{P}_k(a)[d]$) is defined through Rules (**P1**), (**P2**), (**P3**), (**P4**) and (**P5**). Rule (**P1**) considers the case when agent k consumes her permission to perform action a by actually performing it, in this case, the contract reduces to the trivially satisfied one ($\top$). Rule (**P2**) handles the case when an agent other than k performs an action or the action involving b is not the permitted one a, leaving k's permission intact. Rule (**P3**) handles the case when the permission is violated because agent k intended to perform action a, but it was not offered a synchronizing action. Rule (**P4**) considers the case when another agent than k intends to perform an action b (different to a), but it was not offered a synchronizing action. Finally, Rule (**P5**) considers the case when d' time units elapse, with $d' \leqslant d$, then the permission remains, but the permission time decreases by d' time units.

The rules for conditional contracts handle the cases when the condition holds ((**C1**) and (**C2**)), and when it does not ((**C3**) and (**C4**)), resolving the contract to the appropriate branch. The rule (**C5**) considers the case when d' time units pass (with $d' \leqslant d$), in which case the conditional deadline decreases accordingly.

The rules for conjunction and disjunction are structurally identical, since both take the two contracts to evolve concurrently. The difference between the two operators is only exhibited when one of the two operands reduces to $\top$ or $\bot$, which is then handled by the equivalence rules. The first rule (**AO1**) states that the conjunction or disjunction of two formulae evolves along with both operands concurrently.

The second rule (**AO2**) considers the case in which d time units pass for both contracts. Rule (**AO3**) shows the case in which: (i) d time units pass for the first contract, φ, then it evolves to $\top$ and (ii) d' for the second one, ψ, evolving to ψ', with $d' \geqslant d$. Thus, the contracts' conjunction evolves as the second one. (**AO4**) handles the case in which d time units pass for the first contract, φ, then it evolves to φ' and d' time units for the second one, ψ, then it evolves to $\top$, with $d \geqslant d'$, thus the contracts conjunction evolves as the first one. Rules (**AO5**) and (**AO6**) consider the cases in which the first or second contract has been already violated and how the disjunction of both contracts evolve, in an analogous manner as the conjunction.

The rules for reparation and sequential composition are similar. The rules (**V1**) and (**V2**) allow moving along the primary contract when some actions are done or the time passes. There is no need for rules dealing with the recovering from a violation since this is handled by the syntactic equivalence rules. The sequential composition rules (**S1**) and (**S2**) behave in an analogous manner, allowing evolution along with the first contract, with no need for additional rules thanks to the syntactic equivalence rules. It is worth noting that, similar to reparation which fires the second operand on the first (shortest trace) violation, sequential composition fires the second operand on the shortest match of the first operand. Rules (**V3**) and (**S3**) are necessary for time additivity with reparation and sequential composition, respectively.

Example 4 In our running example, we can consider clause φ_4, that is:

$$\varphi_4 ::= \quad (\mathcal{P}_p(\texttt{brd})[90]; \mathcal{P}_p(\texttt{hl})[10]) \blacktriangleright (O_c(\texttt{brd})[90]; O_c(\texttt{hlhld})[20])$$

where the passenger is permitted to board within 90 minutes ($\mathcal{P}_p(\texttt{brd})[90]$) and, then to present the hand-luggage to the staff within 10 minutes ($\mathcal{P}_p(\texttt{hl})[10]$). Therefore, the reparation part of this clause indicates that if the passenger is stopped from boarding or carrying luggage, the airline company is obliged to allow the passenger to board within 90 minutes ($O_c(\texttt{brd})[90]$) and then, to put the passenger's hand luggage in the hold within 20 minutes ($O_c(\texttt{hlhld})[20]$). If 90-time units pass, φ_4 evolves in the following way:

$$\begin{array}{l} \mathcal{P}_p(\texttt{brd})[90]; \mathcal{P}_p(\texttt{hl})[10] \blacktriangleright O_c(\texttt{brd})[90]; O_c(\texttt{hlhld})[20] \\ \overset{90}{\rightsquigarrow} \\ \mathcal{P}_p(\texttt{brd})[0]; \mathcal{P}_p(\texttt{hl})[10] \blacktriangleright (O_c(\texttt{brd})[90]; O_c(\texttt{hlhld})[20] \\ \equiv \top; \mathcal{P}_p(\texttt{hl})[10] \blacktriangleright (O_c(\texttt{brd})[90]; O_c(\texttt{hlhld})[20]) \end{array}$$

The latter equivalence applies by rule 15, since $\mathcal{P}_p(\texttt{brd})[0] \equiv \top$. In turn, rule 9 can be applied to the first part ($\top; \mathcal{P}_p(\texttt{hl})[10] \equiv \mathcal{P}_p(\texttt{hl})[10]$), then $\varphi_4 ::= \mathcal{P}_p(\texttt{hl})[10] \blacktriangleright (O_c(\texttt{brd})[90]; O_c(\texttt{hlhld})[20])$. Thereafter, if 10 time units pass, rule 15 can be applied again ($\mathcal{P}_p(\texttt{hl})[10] \overset{10}{\rightsquigarrow} \mathcal{P}_p(\texttt{brd})[0] \equiv \top$), then $\varphi_4 ::= \top \blacktriangleright (O_c(\texttt{brd})[90]; O_c(\texttt{hlhld})[20])$. And finally, applying rule 11: $\varphi_4 ::= \top \blacktriangleright (O_c(\texttt{brd})[90]; O_c(\texttt{hlhld})[20]) \equiv \top$, then in this case, it is possible to conclude that the contract is satisfied by only applying the congruence relations. □

The wait rules define two possible cases: when d' time units pass, with $d' \leqslant d$, then the time delay decreases by d' time units (Rule (**wait1**)), and when an action is performed (Rule (**wait2**)) the time delay remains intact since (let us recall) actions are instantaneous.

The final rules deal with recursion in a standard manner — by replacing free instances of the recursion variable by the whole recursion formula. Note that since we assume formulae to be closed and recursion guarded, we require no rules for expressions consisting of just a free variable, or to handle unguarded recursion such as $\mathsf{rec}\, x.\, x$.

The following proposition shows that the semantics ensure that any non-trivial contract (i.e. any irreducible contract other than $\top$ and $\bot$) can evolve to any observed action. Furthermore, they evolve in a deterministic manner.

Proposition 3 *Given a contract $\varphi \in \mathcal{C}$:*

1. *One of the following holds: (i) $\varphi \equiv \top$; (ii) $\varphi \equiv \bot$; or (iii) for any $a \in \mathsf{Act}$ and $k \in I$, $\varphi \xrightarrow{a,k}$ and $\varphi \xrightarrow{\overline{(a,k)}}$.*
2. *If $\varphi \xrightarrow{a,k} \varphi_1$ and $\varphi \xrightarrow{a,k} \varphi_2$, then $\varphi_1 \equiv \varphi_2$.*

Proof. The first property follows immediately from the operational semantics. The second follows by structural induction on φ. □

The following proposition shows that the contracts behave coherently with respect to time.

Proposition 4 *Let $\varphi, \varphi', \varphi'' \in \mathcal{C}$ be contracts and $d_1, d_2 \in \mathbb{T}$ be time values. Then, the following properties hold:*

1. *If $\varphi \overset{d_1}{\rightsquigarrow} \varphi'$ and $\varphi \overset{d_1}{\rightsquigarrow} \varphi''$ then $\varphi' \equiv \varphi''$.*
2. *If $\varphi \overset{d_1}{\rightsquigarrow} \varphi' \overset{d_2}{\rightsquigarrow} \varphi''$, then $\varphi \overset{d_1+d_2}{\rightsquigarrow} \varphi''$.*
3. *If $\varphi \overset{d_1+d_2}{\rightsquigarrow} \varphi''$ then there is $\varphi' \in \mathcal{C}$ such that $\varphi \overset{d_1}{\rightsquigarrow} \varphi' \overset{d_2}{\rightsquigarrow} \varphi''$*

Proof. These properties are proved by structural induction. The base cases are trivial, one only needs to take into account that the contracts $O_k(a)[d]$, $\mathcal{F}_k(a)[d]$, $\mathcal{P}_k(a)[d]$ do not transition beyond time d because the contracts are violated (in the case of obligation) or satisfied (in the case for prohibition and permission). □

4.4 Contract Violation

We can now formally define contract violation. First, we define the predicate $\mathsf{vio}(\varphi)$. This predicate will be used to verify if a contract is *currently violated*, which enables us to determine how a system can be monitored with respect to a contract.

Definition 9 *We say that an irreducible contract φ is in a* violated *state, written $\mathsf{vio}(\varphi)$ if and only if the contract has already been violated:*

$$\begin{aligned}
\mathsf{vio}(\top) &\overset{df}{=} \mathit{ff} & \mathsf{vio}(\bot) &\overset{df}{=} \mathit{tt} \\
\mathsf{vio}(\mathcal{P}_k(a)[d]) &\overset{df}{=} \overline{(a,k)} & \mathsf{vio}(O_k(a)[d]) &\overset{df}{=} \mathit{ff} \\
\mathsf{vio}(\mathcal{F}_k(a)[d]) &\overset{df}{=} (a,k) & \mathsf{vio}(\mathsf{wait}(d)) &\overset{df}{=} \mathit{ff} \\
\mathsf{vio}(\varphi;\varphi') &\overset{df}{=} \mathsf{vio}(\varphi) & \mathsf{vio}(\mathsf{rec}\,x.\varphi) &\overset{df}{=} \mathsf{vio}(\varphi) \\
\mathsf{vio}(\mathit{cond}_k(a)[d](\varphi,\varphi')) &\overset{df}{=} \mathit{ff} & & \\
\mathsf{vio}(\varphi \wedge \varphi') &\overset{df}{=} \mathsf{vio}(\varphi) \vee \mathsf{vio}(\varphi') & & \\
\mathsf{vio}(\varphi \vee \varphi') &\overset{df}{=} \mathsf{vio}(\varphi) \wedge \mathsf{vio}(\varphi') & & \\
\mathsf{vio}(\varphi \blacktriangleright \varphi') &\overset{df}{=} \mathsf{vio}(\varphi) \wedge \mathsf{vio}(\varphi') & &
\end{aligned}$$

Since syntactical equivalences would remove any zero time windows (i.e. $d = 0$), the above definition covers only when $d > 0$.

The two first cases for the trivially satisfied and violated contracts are straightforward. In the case of permission being currently in force, we flag a violation if the party holding the permission wants to perform the action but is not offered a synchronizing action. In case of an obligation, a violation can only occur after the time has expired ($d = 0$), but this case is already defined because of the syntactical equivalence $O_k(a)[0] \equiv \bot$. Let us note that an obligation to perform an action within a (non-zero) time frame is never in violation at this instant since there is still time to perform the action and fulfil the obligation.

In the case of a reparation $\mathsf{vio}(\varphi \blacktriangleright \varphi')$, a violation can only occur, if both φ and φ' are violated. In the case of $\mathsf{vio}(\mathrm{cond}_k(a)[d](\varphi,\varphi'))$, whether the action a or any other action is observed the violation is always false since the conditional contract only defines how the contract will behave (as φ or φ'). In the case of sequential composition $\mathsf{vio}(\varphi;\varphi')$, an immediate violation must occur on the first operand (since $\top;\varphi$ would have been reduced to φ), and it is thus defined as $\mathsf{vio}(\varphi)$. In the case of $\mathsf{wait}(d)$, the violation is always false, since it depicts a time delay, then an immediate violation is false. Finally, the definition $\mathsf{vio}(\mathsf{rec}\,x.\varphi) = \mathsf{vio}(\varphi)$ is well-formed since recursion is always assumed to be guarded.

Lemma 1 *For any contract $\varphi \in \mathcal{C}$, $\mathsf{vio}(\varphi) \dashv\models \mathit{tt}$ if and only if $\varphi \equiv \bot$.*

Proof. The proof uses structural induction on φ. The only non-trivial case being when $\varphi = \varphi_1 \wedge \varphi_2$, in which case we use Proposition 1. □

4.5 Contracts Acting on Systems

We can now define how contracts evolve alongside a system, and what it means for a system to satisfy a contract.

Definition 10 *Given a contract $\varphi \in \mathcal{C}$ with a set of actions Act' and a system $\mathcal{A}$, we define the semantics of $\varphi \| \mathcal{A}$ — the combination of the system with the contract — with alphabet Act with $\mathsf{Act}' \subseteq \mathsf{Act}$ through the following rules:*

(M1)	$\dfrac{\varphi \xrightarrow{a,k} \varphi',\ \mathcal{A} \xrightarrow{a,s} \mathcal{A}'}{\varphi \parallel \mathcal{A} \Rightarrow \varphi' \parallel \mathcal{A}'} k \in s$
(M2)	$\dfrac{\varphi \xrightarrow{\overline{(a,k)}} \varphi',\ \mathcal{A} \models \langle a, \overline{k} \rangle}{\varphi \parallel \mathcal{A} \Rightarrow \varphi' \parallel \mathcal{A}}$
(M3)	$\dfrac{\mathcal{A} \xrightarrow{a,s} \mathcal{A}'}{\varphi \parallel \mathcal{A} \Rightarrow \varphi \parallel \mathcal{A}'} a \notin \mathsf{Act}'$
(M4)	$\dfrac{\begin{array}{c}\mathcal{A} \overset{d}{\rightsquigarrow} \mathcal{A}',\ \varphi \overset{d}{\rightsquigarrow} \varphi', \\ \forall d' < d \cdot \text{ if } \mathcal{A} \overset{d'}{\rightsquigarrow} \mathcal{A}'' \text{ and} \\ \varphi \overset{d'}{\rightsquigarrow} \varphi'' \text{ then } \mathcal{A}'' \not\models \mathsf{vio}(\varphi'')\end{array}}{\varphi \parallel \mathcal{A} \Rightarrow \varphi' \parallel \mathcal{A}'}$

Rule (**M1**) and (**M2**) handles synchronization between the contract and the system. If an action a performed by the system is of interest to the contract, the contract evolves alongside the system ((**M1**)), if the contract allows an agent to perform an action but only agent k (and no other agent) is willing to engage in the action, then only the contract evolves ((**M2**)). Rule (**M3**) handles actions on the system in which the contract is not interested in. Finally, rule (**M4**) ensures that time cannot skip over a violation.

Definition 11 *Let $\mathcal{A}$ be a system and $\varphi \in \mathcal{C}$ be a contract.*

- *System $\mathcal{A}$ can* breach φ*, written* $\mathsf{breach}(\mathcal{A}, \varphi)$*, if there exists a computation that leads to a violation of the contract: for some $n \geqslant 0$ and contracts φ_0 till φ_n such that:*

$$\varphi \parallel \mathcal{A} = \varphi_0 \parallel \mathcal{A}_0 \Rightarrow \ldots \varphi_{n-1} \parallel \mathcal{A}_{n-1} \Rightarrow \varphi_n \parallel \mathcal{A}_n,$$

  and $\mathcal{A}_n \models \mathsf{vio}(\varphi_n)$.
- *System $\mathcal{A}$ may* fulfil φ*, written* $\mathit{fulfill}(\mathcal{A}, \varphi)$*, if there exists a computation of the system that fulfils the contract: for some $n \geqslant 0$ and contracts φ_0 till φ_n:*

$$\varphi \parallel \mathcal{A} = \varphi_0 \parallel \mathcal{A}_0 \Rightarrow \ldots \varphi_{n-1} \parallel \mathcal{A}_{n-1} \Rightarrow \varphi_n \parallel \mathcal{A}_n\ ,$$

  and $\mathcal{A} \not\models \mathsf{vio}(\varphi_k)$ for $0 \leqslant k < n$, and $\varphi_n \equiv \top$.

Note that there are contracts that may never be fulfilled. An example of such a contract is $\varphi = \mathsf{rec}\, x.[a,k,\infty](\bot,\infty)$, which may never be fulfilled since there are no transitions from this contract leading to $\top$. Nevertheless, if agent k never performs action a, then neither is the contract broken.

5 Refinement

We now define two notions of contract refinement ($\preccurlyeq_\bot$ and $\preccurlyeq_\top$). Intuitively $\preccurlyeq_\bot$ relates two contracts $\varphi, \psi \in \mathcal{C}$ (i.e. $\varphi \preccurlyeq_\bot \psi$) if any system which can breach contract φ, can also breach contract ψ. The meaning of $\preccurlyeq_\top$ is its dual: if $\varphi \preccurlyeq_\top \psi$ then any system which can fulfil φ can also fulfil ψ. Both notions are based on simulation techniques, defined in a co-inductive fashion.

Definition 12 *Let $\varphi, \psi \in \mathcal{C}$ and $R \subseteq \mathcal{C} \times \mathcal{C}$, we say that R is a* $\bot$-simulation *relation iff whenever $(\varphi, \psi) \in R$ the following conditions hold:*

(i) $\mathsf{vio}(\varphi) \models \mathsf{vio}(\psi)$.

(ii) If $\varphi \overset{d}{\rightsquigarrow} \varphi'$ *then*

  a. there is $d' \leqslant d$ such that $\psi \overset{d'}{\rightsquigarrow} \bot$, or

  b. there is $\psi' \in \mathcal{C}$ and $\psi \overset{d}{\rightsquigarrow} \psi'$ and $(\varphi', \psi') \in R$.

(iii) If $\varphi \xrightarrow{\alpha} \varphi'$ *then there exists $\psi' \in \mathcal{C}$ and $\psi \xrightarrow{\alpha} \psi'$ and $(\varphi', \psi') \in R$.*

We say $\varphi \preccurlyeq_\bot \psi$ if there is a $\bot$-simulation relation R such that $(\varphi, \psi) \in R$.

Definition 13 *Let $\varphi, \psi \in \mathcal{C}$ and $R \subseteq \mathcal{C} \times \mathcal{C}$, we say that R is a* $\top$-simulation *relation iff whenever $(\varphi, \psi) \in R$, the following conditions hold:*

(i) If $\varphi \equiv \top$ then $\psi \equiv \top$.

(ii) If $\mathsf{vio}(\psi) \models \mathsf{vio}(\varphi)$.

(iii) If $\varphi \overset{d}{\rightsquigarrow} \varphi'$ *then*

  a. there is $d' \leqslant d$ such that $\psi \overset{d'}{\rightsquigarrow} \top$, or

  b. there is $\psi' \in \mathcal{C}$ and that $\psi \overset{d}{\rightsquigarrow} \psi'$ and $(\varphi', \psi') \in R$.

(iv) If $\varphi \xrightarrow{\alpha} \varphi'$ *then there is $\psi' \in \mathcal{C}$ such that $\psi \xrightarrow{\alpha} \psi'$ and $(\varphi', \psi') \in R$.*

We say $\varphi \preccurlyeq_\bot \psi$ if there is a $\bot$-simulation relation R such that $(\varphi, \psi) \in R$.

Lemma 2 *The relation $id = \{(\varphi, \varphi) \mid \varphi \in \mathcal{C}\}$ is both a $\bot$-simulation relation and a $\top$-simulation relation.*

Proof. It is immediate from the definitions. □

Lemma 3 *Let R_1 and R_2 be $\bot$-simulation relations (respectively $\top$-simulation). Then, their composition $R_1 \circ R_2$ is also a $\bot$-simulation relation (respectively $\top$-simulation).*

Proof. The proof is simple from the definitions. □

Proposition 5 *The relations $\preccurlyeq_\bot$ and $\preccurlyeq_\top$ are reflexive and transitive.*

Proof. This proposition is immediate from Lemmas 2 and 3. □

Consider the following example illustrating the use of these definitions.

Example 5

$$\mathsf{wait}(3) \preccurlyeq_\bot \mathcal{P}_k(a)[5]$$
$$\mathsf{wait}(3); \bot \not\preccurlyeq_\bot \mathcal{P}_k(a)[5]; \bot$$
$$\mathcal{P}_k(a)[3] \preccurlyeq_\top \mathsf{wait}(3)$$
$$\mathcal{P}_k(a)[3] \triangleright O_l(b)[2] \not\preccurlyeq_\top \mathsf{wait}(3) \triangleright O_l(b)[2]$$
$$\mathsf{wait}(5) \preccurlyeq_\bot O_k(a)[6]$$
$$\mathsf{wait}(5) \wedge \mathsf{wait}(7) \preccurlyeq_\bot O_k(a)[6] \wedge \mathsf{wait}(7)$$

It is not difficult to formally verify the correctness of these orderings. For instance, consider $\mathsf{wait}(3)$ and $\mathcal{P}_k(a)[5]$ — the former cannot be violated, whilst the latter can be violated by any system that does not allow agent k to perform action a within 5 units of time, which ensures that the simulation holds. Now consider $\mathsf{wait}(3) \preccurlyeq_\bot \mathcal{P}_k(a)[5]$, we can put both contracts in the context of the continuation operator to follow up with $\bot$. While $\mathsf{wait}(3);\bot$ cannot be fulfilled after 3 units of time whatever the system does, in the case of $\mathcal{P}_k(a)[5];\bot$, if the system allows agent k to perform a after 3 units of time but agent k does not perform the action, the contract is not broken yet. Regarding the $\preccurlyeq_\top$ relation, dual reasoning can be applied, whilst the other relations can be similarly reasoned about. □

Since the relations are preorders, for each of them we have an equivalence relation. However, we can prove that these relations are, in fact, equivalent.

Proposition 6 *The two equivalence relations* $\backsim_\top = \preccurlyeq_\top \cap \preccurlyeq_\top{}^{-1}$ *and* $\backsim_\bot = \preccurlyeq_\bot \cap \preccurlyeq_\bot{}^{-1}$*, are equal to each other:* $\backsim_\top = \backsim_\bot$.

Proof. In order to prove $\backsim_\bot \subseteq \backsim_\top$ we have to prove $\backsim_\bot \subseteq \preccurlyeq_\bot$ and $\backsim_\bot \subseteq \preccurlyeq_\bot{}^{-1}$. Both proofs are symmetrical, so let us prove the first. It is sufficient to prove that $\backsim_\bot$ is a $\top$-simulation relation. Consider $\varphi, \psi \in \mathcal{C}$ such that $\varphi \backsim_\bot \psi$ — we must prove the conditions of Definition 13, with the only non-trivial one being condition i. Assume $\varphi \equiv \top$. Since $\varphi \backsim_\bot \psi$, we deduce $\mathsf{vio}(\psi) = ff$. If $\psi \not\equiv \top$, then by Proposition 3, for any possible α there must exist ψ' such that $\psi \xrightarrow{\alpha} \psi'$. Again, since $\varphi \backsim_\bot \psi$ we obtain that for any α, there must exist φ' such that $\top = \varphi \xrightarrow{\alpha} \varphi'$, which is impossible. We can thus conclude that $\psi \equiv \top$. □

Given their equivalence, we can define the simulation equivalence of contracts as either of the two equivalence relations.

Definition 14 *We define the* simulation equivalence relation *as* $\backsim \stackrel{df}{=} \preccurlyeq_\top \cap \preccurlyeq_\top{}^{-1}$

Consider the $\bot$ simulation: if two contracts are related $\varphi \preccurlyeq_\bot \psi$, then the violations identified by φ are also identified by ψ.

Theorem 1 *Let $\mathcal{A}$ be a system and $\varphi, \psi \in \mathcal{C}$ be contracts, such that $\varphi \preccurlyeq_\bot \psi$. Then, if $\mathcal{A}$ violates φ, it also violates ψ:* $\mathsf{breach}(\mathcal{A}, \varphi) \Rightarrow \mathsf{breach}(\mathcal{A}, \psi)$.

Proof. Since $\varphi \preccurlyeq_\bot \psi$, then there exists a simulation contract relation R, such that $(\varphi, \psi) \in R$. On the other hand, since $\mathsf{breach}(\mathcal{A}, \varphi)$ holds, there exists a sequence of transitions

$$\varphi \parallel \mathcal{A} = \varphi_0 \parallel \mathcal{A}_0 \Rightarrow \ldots \varphi_n \parallel \mathcal{A}_n = \varphi' \parallel \mathcal{A}'$$

where $n \geqslant 0$, such that $\mathsf{breach}(\mathcal{A}', \varphi')$. By simulating φ, we can build a computation beginning with the contract $\psi_0 = \psi$:

$$\psi \parallel \mathcal{A} = \psi_0 \parallel \mathcal{A}_0 \Rightarrow \ldots \psi_m \parallel \mathcal{A}_m = \psi' \parallel \mathcal{A}'$$

such that $m \leqslant n$, $(\varphi_k, \psi_k) \in R$ for $0 \leqslant k < m$, and $\mathsf{breach}(\mathcal{A}', \psi_m)$. Let us proceed by induction. If $n = 0$ the proof is immediate, so let us consider the inductive case $n > 0$. Let us consider the first transition. There are four cases according to the rules of the system transitions (Definition 1):

Rules M1 and M2. $\varphi_0 \xrightarrow{\alpha} \varphi_1$. Since $(\varphi_0, \psi_0) \in R$, then there is a contract ψ_1 such that $\psi_0 \xrightarrow{\alpha} \psi_1$ and $(\varphi_1, \psi_1) \in R$. Therefore, we obtain that we have the computation $\psi_0 \parallel \mathcal{A}_0 \Rightarrow \psi_1 \parallel \mathcal{A}_1$. Then we obtain the result by induction.

Rule M3. This is trivial because the contract is not involved.

Rule M4. $\varphi_0 \overset{d}{\rightsquigarrow} \varphi_1$ then either:

1. There exists $d' < d$ such that $\psi_0 \overset{d}{\rightsquigarrow} \bot$. In this case we obtain $\psi \parallel \mathcal{A} \Rightarrow \bot \parallel \mathcal{A}'$.
2. There exists ψ_1 such that $\psi_0 \overset{d}{\rightsquigarrow} \psi_1$ and $(\varphi_1, \psi_1) \in R$. If there were $0 < d' < d$ such that $\psi \overset{d'}{\rightsquigarrow} \psi'$, $\mathcal{A} \overset{d'}{\rightsquigarrow} \mathcal{A}'$, and $\mathcal{A}' \models \mathsf{vio}(\psi')$, then we obtain the result immediately. Otherwise $\psi \parallel \mathcal{A} \Rightarrow \psi_1 \parallel \mathcal{A}_1$ and we obtain the result by induction.

Finally, if $m < n$ we have found the computation $\psi \parallel \mathcal{A} \Rightarrow^* \bot \parallel \mathcal{A}'$. Otherwise $(\varphi_n, \psi_n) \in R$, then $\mathsf{vio}(\varphi_n) \models \mathsf{vio}(\psi_n)$, and by definition $\mathcal{A}_n \models \mathsf{vio}(\psi_n)$. □

Now let us prove the corresponding property of $\top$ simulated contract. If two contracts are related $\varphi \preccurlyeq_\top \varphi'$, and if φ can be fulfilled by a system, then φ' is also fulfilled by the same system.

Theorem 2 *Let $\mathcal{A}$ be a system and $\varphi, \psi \in \mathcal{C}$ be contracts such that $\varphi \preccurlyeq_\top \psi$. Then, if $\mathcal{A}$ can fulfil φ, it can also fulfil ψ:* $\mathsf{fulfill}(\mathcal{A}, \varphi) \Rightarrow \mathsf{fulfill}(\mathcal{A}, \psi)$.

Proof. The proof of this theorem is very similar to the previous one. The inductive cases for rules **M1**, **M2** and **M3** are similar: We only have to verify $\mathcal{A}_k \nVdash \mathsf{vio}(\psi_k)$, which is immediate since $\mathsf{vio}(\psi_k) \models \mathsf{vio}(\varphi_k)$. The case **M4** is slightly different; so let us assume $\varphi_0 \xrightarrow{d} \varphi_1$. First let us suppose that there exists $d' < d$ such that $\psi_0 \xrightarrow{d'} \psi'$, $\mathcal{A} \overset{d'}{\rightsquigarrow} \mathcal{A}'$ and $\mathcal{A}' \models \mathsf{vio}(\psi')$. Due to Proposition 4 and the definition of $\top$ simulation contract, there exists φ' such that $\varphi_0 \xrightarrow{d'} \varphi'$ with $(\varphi', \psi') \in R$. Therefore $\mathsf{vio}(\psi') \models \mathsf{vio}(\varphi')$ and then the transition $\varphi \parallel \mathcal{A} \Rightarrow \varphi_1 \parallel \mathcal{A}$ is not possible. Now, since $(\varphi_0, \psi_0) \in R$ there are two cases:

1. There exists $d' < d$ such that $\psi \overset{d'}{\rightsquigarrow} \top$, so in this case we have found the computation $\psi_0 \parallel \mathcal{A} \overset{d'}{\rightsquigarrow} \top \parallel \mathcal{A}$.

2. There exists ψ_1 such that $\psi_0 \overset{d}{\rightsquigarrow} \psi_1$ and $(\varphi_1, \psi_1) \in R$. If $\varphi_1 \equiv \top$ then $\psi_1 \equiv \top$ and we have found the required computation. Otherwise, $\mathsf{vio}(\psi_1) \models \mathsf{vio}(\varphi_1)$ and then $\mathcal{A}_1 \not\models \mathsf{vio}(\psi_1)$, so we obtain the result by induction. □

Finally, in this section we are going to show important properties of the relations $\preccurlyeq_\top$ and $\preccurlyeq_\bot$. First, let us show that $\top$ and $\bot$ are *the best* contracts in their respective relations $\preccurlyeq_\top$ and $\preccurlyeq_\bot$. Then, as $\varphi \wedge \top \equiv \varphi$ and $\varphi \vee \bot \equiv \varphi$, it is important to show $\varphi \wedge \varphi' \preccurlyeq_\top \varphi$ and $\varphi \vee \varphi' \preccurlyeq_\bot \varphi$.

Proposition 7 *For any $\varphi, \varphi' \in \mathcal{C}$, the following hold:*

1. $\varphi \preccurlyeq_\top \top$
2. $\varphi \preccurlyeq_\bot \bot$
3. $\varphi \vee \varphi' \preccurlyeq_\bot \varphi$
4. $\varphi \wedge \varphi' \preccurlyeq_\top \varphi$
5. $\varphi \backsim \varphi \vee \varphi$
6. $\varphi \backsim \varphi \wedge \varphi$

Proof. Statements *1* and *2* follow from the definitions and Lemma 1. For *3* we have to check that $R_\bot = \{(\varphi \vee \varphi', \varphi) \mid \varphi, \varphi' \in \mathcal{C}\}$ is a $\bot$ simulation contract. While for *4* we have to check that $R_\top = \{(\varphi \wedge \varphi', \varphi) \mid \varphi, \varphi' \in \mathcal{C}\}$ is a $\top$ simulation contract. For *5* and *6* it is easy to check that the relations $R_\vee = \{(\varphi, \varphi \vee \varphi) \mid \varphi \in \mathcal{C}\}$, $R'_\vee = \{(\varphi \vee \varphi, \varphi) \mid \varphi \in \mathcal{C}\}$, $R_\wedge = \{(\varphi, \varphi \wedge \varphi) \mid \varphi \in \mathcal{C}\}$, and $R'_\wedge = \{(\varphi \wedge \varphi, \varphi) \mid \varphi \in \mathcal{C}\}$ are respectively both, $\top$ simulation contracts and $\bot$ simulation contracts. □

Let us show cases in which the relations act as congruences.

Proposition 8 *For any $\varphi, \varphi', \psi, \psi' \in \mathcal{C}$, the following hold:*

If $\varphi' \preccurlyeq_\bot \varphi$ and $\psi' \preccurlyeq_\bot \psi$:	*If $\varphi' \preccurlyeq_\top \varphi$ and $\psi' \preccurlyeq_\top \psi$:*
$\bot.1$ $\varphi' \triangleright \psi' \preccurlyeq_\bot \varphi \triangleright \psi$	$\top.1$ $\varphi'; \psi' \preccurlyeq_\top \varphi; \psi$
$\bot.2$ $\varphi' \wedge \psi' \preccurlyeq_\bot \varphi \wedge \psi$	$\top.2$ $\varphi' \wedge \psi' \preccurlyeq_\top \varphi \wedge \psi$
$\bot.3$ $\varphi' \vee \psi' \preccurlyeq_\bot \varphi \vee \psi$	$\top.3$ $\varphi' \vee \psi' \preccurlyeq_\top \varphi \vee \psi$
$\bot.4$ $cond_k(a)[d](\varphi', \psi') \preccurlyeq_\bot cond_k(a)[d](\varphi, \psi)$	$\top.4$ $cond_k(a)[d](\varphi', \psi') \preccurlyeq_\top cond_k(a)[d](\varphi, \psi)$

Proof. For all the cases consider the relations:

$$R_{(\mathrm{op},\mathrm{rel})} = \preccurlyeq_{\mathrm{rel}} \cup \{(\varphi \,\mathrm{op}\, \psi, \varphi' \,\mathrm{op}\, \psi') \mid \varphi, \varphi', \psi, \psi' \in \mathcal{C}, \varphi \preccurlyeq_{\mathrm{rel}} \varphi', \psi \preccurlyeq_{\mathrm{rel}} \psi'\}$$

where $\mathrm{op} \in \{;, \triangleright, \wedge, \vee, \mathrm{cond}_l(a)[d](\cdot,\cdot)\}$ and $\mathrm{rel} \in \{\top, \bot\}$. In all cases we have to prove that $R_{(\mathrm{op},\mathrm{rel})}$ is a rel simulation. Also in all cases we are going to consider $(\chi, \chi') \in R_{(\mathrm{op},\mathrm{rel})}$ and to prove that $(\chi, \chi') \in \preccurlyeq_{\mathrm{rel}}$. If $(\chi, \chi') \in \preccurlyeq_{\mathrm{rel}}$ there is nothing to prove, so we consider that $\chi = \varphi \,\mathrm{op}\, \psi$, $\chi' = \varphi' \,\mathrm{op}\, \psi'$, $\varphi \preccurlyeq_{\mathrm{rel}} \varphi'$ and $\psi \preccurlyeq_{\mathrm{rel}} \psi'$. In all cases we have to check the conditions on the corresponding relation in Definitions 12 and 13. □

Theorem 3 *$\backsim$ is a congruence.*

Proof. This follows from the previous proposition and Proposition 6. □

6 Conclusions

The calculus Themulus allows us to reason about contracts with time constraints independent of the systems on which they are applied to. In order to achieve this, we have introduced a notion of similarity between contracts, which takes into account predicates over system states, and shows how these semantics can be used for runtime verification of contracts.

There are various research directions we intend to explore. From a practical perspective, we will be looking into automated runtime verification of contracts, and looking at how this scales up with more complex contracts. From a theoretical perspective, there are various questions we have yet to explore — from identifying conflicts in our contract language, to looking at automated synthesis of the strongest contract satisfied by a given system (analogous to the weakest-precondition) and synthesis of the weakest system satisfying a given contract.

One application arising from runtime monitoring was that of runtime enforcement, where starting from a specification, algorithmic machinery is synthesized to ensure that the system under scrutiny does not violate the specification e.g. by delaying or injecting events. In particular, there is a body of work on runtime enforcement of timed properties e.g. [Falcone et al., 2016] which could offer insight on how our work can be extended to build contract enforcement engines, a notion that has not been widely explored in the deontic logic world.

References

[Asarin et al., 2012] Asarin, E., Mysore, V., Pnueli, A., and Schneider, G. (2012). Low dimensional hybrid systems - decidable, undecidable, don't know. *Inf. Comput.*, 211:138–159.

[Azzopardi et al., 2014] Azzopardi, S., Pace, G. J., and Schapachnik, F. (2014). Contract automata with reparations. In *Legal Knowledge and Information Systems - JURIX: The Twenty-Seventh Annual Conference, Jagiellonian University, Krakow, Poland, 10-12 December*, pages 49–54.

[Belnap and Perloff, 1993] Belnap, N. and Perloff, M. (1993). In the realm of agents. *Ann. Math. Artif. Intell.*, 9(1-2):25–48.

[Bocchi et al., 2014] Bocchi, L., Yang, W., and Yoshida, N. (2014). Timed multiparty session types. In *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings*, pages 419–434.

[Cambronero et al., 2017] Cambronero, M., Llana, L., and Pace, G. J. (2017). A calculus supporting contract reasoning and monitoring. *IEEE Access*, 5:6735–6745.

[Chaochen et al., 1991] Chaochen, Z., Hoare, C. A. R., and Ravn, A. P. (1991). A calculus of durations. *Inf. Process. Lett.*, 40(5):269–276.

[Falcone et al., 2016] Falcone, Y., Jéron, T., Marchand, H., and Pinisetty, S. (2016). Runtime enforcement of regular timed properties by suppressing and delaying events. *Sci. Comput. Program.*, 123:2–41.

[Fenech et al., 2009] Fenech, S., Pace, G. J., Okika, J. C., Ravn, A. P., and Schneider, G. (2009). On the specification of full contracts. *Electr. Notes Theor. Comput. Sci.*, 253(1):39–55.

[Georg Henrik Von Wright, 1951] Georg Henrik Von Wright (1951). Deontic Logic. *Mind*, 60(237):1–15.

[Governatori and Milosevic, 2005] Governatori, G. and Milosevic, Z. (2005). Dealing with contract violations: formalism and domain specific language. In *EDOC Enterprise Computing Conference, Ninth IEEE International*, pages 46–57. IEEE Computer Society.

[Horty, 2001] Horty, J. F. (2001). *Agency and Deontic Logic*. Oxford University Press.

[Jeremaes et al., 1986] Jeremaes, P., Khosla, S., and Maibaum, T. (1986). A modal (action) logic for requirements specification. *Software Engineering*, 86:278–294.

[Khosla, 1988] Khosla, S. (1988). *System Specification: A Deontic Approach*. PhD thesis, Imperial College of Science and Technology, University of London.

[Milner, 1999] Milner, R. (1999). *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press.

[Newman, 1942] Newman, M. H. A. (1942). On theories with a combinatorial definition of" equivalence". *Annals of mathematics*, pages 223–243.

[Pace and Schapachnik, 2012] Pace, G. J. and Schapachnik, F. (2012). Contracts for Interacting Two-Party Systems. In *FLACOS'12*, volume 94 of *ENTCS*, pages 21–30.

[Yi, 1991] Yi, W. (1991). CCS + time = an interleaving model for real time systems. In *Automata, Languages and Programming, 18th International Colloquium, ICALP91, Madrid, Spain, July 8-12, 1991, Proceedings*, pages 217–228.