Themulus: A Timed Contract-Calculus.

Technical Report TR-01-20

Alberto Aranda García María-Emilia Cambronero Christian Colombo Luis Llana Gordon J. Pace

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 [1] 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 [2]. 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 [3]. 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 [4], deontic STIT logic [5], [6], Business Contract Language [7], contract automata [8]), in which, for instance, a permission is parametrized by (i) the agent which is to be permitted to perform an action or be in a particular state; and (ii) the agent which is bound to provide that permission.

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 appeared in this area. In much of the literature, however, contract comparison is still defined by quantifying over all possible behaviour of the systems, making reasoning about contracts directly dependant on contract satisfaction and violation predicates parametrized by the behaviour they are regulating.

Orthogonal to these issues is that of the notion of time in contracts. From work in temporal logics, one can categorize such logics into a number of broad 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 [9]. 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. We discuss this further in Section 2. Our approach is an interval logic one, taking the approach adopted by real time logics such as duration calculus [10], 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 [11] in which, only temporal sequentiality of events was handled. In this paper, we:

- present a real time extension, Themulus, of the calculus to reason about contracts abstracting away the agents' behaviour in the simulation relation and considering deontic modalities and conditions with time constraints;
- give an operational view of contracts, using notions from process calculi to model the notion of contracts, and enabling their analysis;
- use simulation techniques from process calculi to reason about contracts at an operational level, allowing the simplification of contracts and reasoning about nondeterminism (in contracts);
- present an algorithm for the runtime monitoring of contracts written in Themulus; and
- report on a case study which has been successfully

- *M. Emilia Cambronero is with the Department of Computer Science, University of Castilla-La Mancha, Albacete, Spain. E-mail: {MEmilia.Cambronero}@uclm.es*
- *Luis Llana is with the Department of Computer Science, University Complutense of Madrid, Madrid, Spain. E-mail: {llana}@ucm.es*
- *Alberto Aranda, Christian Colombo and Gordon J. Pace are with the Department of Computer Science, University of Malta, Malta. E-mail: alb.aranda@gmail.com, Christian.Colombo@um.edu.mt, Gordon.Pace@um.edu.mt*

runtime verified through our novel implementation of the calculus.

The paper is organised as follows. In section 2, we compare our approach to related work, after which we present a running example (section 3) used throughout the paper to clarify concepts. Then, the notation we will use to formalize our notions is presented in section 4. We then present our timed contract calculus in section 5 and formalize the notion of refinement of contracts in section 6. We show how the calculus can be applied to a case study of an airport plane boarding system agreement between a passenger, security airport staff, and the airline company in section 7. Furthermore, we show how we can transform contracts written in our calculus into runtime monitors and give the empirical results of our implementation in section 8. We finally conclude in section 9 with some conclusions and possible lines of future work.

2 Related Work

There is a long history of contract formalization in terms of deontic logics. The time-aware calculus we present in this paper is based on the contract calculus we presented in [11], in which contracts were only aware of event sequentiality but not their actual timing. Putting aside the timed aspect of the calculus, our approach has three important features: (i) deontic modalities are explicitly tagged by the party involved; (ii) the use of operational semantics allows us to compare contracts beyond the trace level, using the standard notion of simulation; and (iii) interaction between parties is an important aspect of the calculus, since we need to take into account whether the parties allow each other to satisfy the contract. A detailed comparison between the untimed part of the calculus and other approaches in the literature can be found in [11]. In this paper we will focus solely on related work from the literature formalizing time in contracts. The way we augment the calculus with time shares much with how real-time is typically added to process calculi e.g. timed CSP [12] and timed CCS [13] — we enrich the operational semantics to allow for time taking transitions. In this manner, we can compare contracts also with respect to their behaviour over time.

Although time plays an important role in contracts, there is limited work formalizing the notions behind explicitly timed contracts. Even allowing for temporal modalities, many deontic logics and contract languages e.g. [14], [15], [16] limit their notion of time to one of temporal ordering of events e.g. the obligation to pay immediately *after* using a service (i.e. a trace which contains the subtrace $\langle useService, somethingElse \rangle$ will result in a violation), or prohibition from borrowing a fourth book from a library without returning any of the ones already in your possession (i.e. any trace containing a subtrace which has four instances of *borrowBook* without intermediate *returnBook*).

Still, one can find a number of logics considering explicit time in contracts e.g. [17], [18], [19]. In [17], a graph-based representation of contracts is used to represent deontic clauses of different signatories and the absolute and relative timing constraints associated to them. Semantics based on timed automata extended with information regarding the satisfaction and violation of clauses in order to represent different deontic modalities is shown, but the formalism considers the parties independently, and lacks an operational semantics.

Broersen et al. [18] study a dyadic modal operator, which covers the notion of *being obliged to obey a condition before another condition occurs*. For this purpose, they use logic CTL and try to extend it with a set of violation constants. By considering deadlines as explicit events, they effectively deal with time using just a notion of ordering, and thus cannot deal with relative deadlines without additional logical overheads (explicitly introducing rules such as the ordering of two sequential one second deadlines with respect to a three second interval). Therefore, they define a simple semantics for obligations with deadlines in terms of branching time models. This work is mainly focused on time associated with obligations, and it does not consider time for the other deontic operators.

Cole et al. [19] present a case study of a conference programme to illustrate how policies (also covering obligations, prohibitions and permissions) might be syntactically formulated and refined alongside the refinement of the system specification. They model timed actions and events, but they do not give a formal semantics for this purpose.

One of the richer real-time deontic logics appears in Marjanovic et al. [20]. The logic provides various ways in which temporal constraints can be added within contracts, including the verification of constraints over conjunctions of contract clauses. However, the logic lacks notions such as contract sequentiality and reparation which are able to handle in our logic.

The closest to our approach of building a calculus to reason about events is that used in the event-calculus [21] and its formalization in XML [22]. The event-calculus gives a logical framework to reason about contracts in a trace-based manner. Although, in the original calculus, deontic modalities are based on time points (as opposed to over temporal intervals), the use of quantification over temporal variables allows for the lifting of these point-wise modalities to intervals, albeit in an *ad hoc* manner. Later work [23], [24], [25] introduced the notions of different forms of obligations with deadlines, which correspond closely to our notion of obligation and reparation in this paper. There is related work in formalizing such obligations with deadlines using defeasible logic in [26], [27], extended with reparations in [28]. Finally, the notion of deadlines was extended from obligations to permissions in [29] which is also similar to our notion of time-bounded permissions. The main difference between this body of work and our approach is not in the expressiveness of the contract language or the deontic modalities themselves, but rather in the formalization used. Our approach takes a simulation-based approach which allows for comparison between contracts even in the presence of non-determinism, and gives us an inherent notion of contract refinement. Also, the notion of refuted actions (actions which a party does not allow happening by not providing its part of the handshake) is also crucial in the presence of interactive parties, and is handled implicitly in the simulation relation without having to introduce additional modalities such as intention (or the attempt) to perform an action.

Simulation techniques have been widely studied in the

context of reactive systems. One of the most interesting properties of the simulation semantics is that they can be efficiently computed [30], [31]. The algorithm to compute the simulation semantics can also reduce the number of states of the systems [32]. This reduced version can be used to efficiently perform other formal techniques such as model checking and monitoring.

In the field of runtime verification, there is a sizeable body of work addressing the runtime monitoring, verification and enforcement, both for analogue signals (e.g. [33], [34]) and, closer to our work, event-based ones (e.g. [35], [36], [37]). However, the distinction from the work we present in this paper, is that the specification languages used in these approaches and tools (e.g. variants of LTL, timed regular expressions, variants of timed automata) do not carry deontic modalities, thus making reasoning about the contract (specification in their case) itself as a first class entity (e.g. to reason about reparations, contract conflicts, contract transformation, etc.) more burdensome.

3 Running Example

In the rest of the paper, we will illustrate our logic and results using a common running example, that of a plane boarding system as inspired by [38]. In this section we present this simple use case — an agreement between the passenger and airline company , regulating the plane boarding process, from check-in till the flight, including time constraints. This use case is a simplified version of the real-life case study presented in Section 7 to evaluate the logic, and which is 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.
6) 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.

Table 1 show the agreement as a list of the obligations, permissions, and prohibitions that have been inferred from the natural language description of the rules.

We will be using clauses from this small example to illustrate the formal notions we introduce in the rest of the paper.

4 Background and Notation

Contracts regulate the behaviour of a number of agents, or parties running in parallel. In this section we present the notation we will use 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 a number of 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 $\mathcal{I}$ i.e. the system will be of the form $||_{i\in\mathcal{I}} A_i$. 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 in the agents will be assumed to consist of actions over an alphabet Act, and the agents' behaviour will be assumed to consist of (i) a relation indicating how their state changes whenever such an action occurs; and (ii) a relation indicating how they change with the passing of time. Time will be taken to range over the non-negative reals: $\mathbb{T} = \mathbb{R}^+$. Agents semantics are thus assumed to be represented as *timed labelled transition systems*:

- $A \xrightarrow{a} A'$, for $a \in$ Act, indicates that agent A changes to A' upon performing action a. As it is usual in process algebrae [13], the execution of actions do not consume time. The transition $A \xrightarrow{a}\!\!\!\!\!\!\rightarrow$ indicates that agent A cannot perform action 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 $a \in$ Act); or (ii) $A \overset{d}{\rightsquigarrow} A'$ (for some $d > 0 \in \mathbb{T}$). We also assume the following properties of the time transition relation:

- *Time determinism*: If $A \overset{d}{\rightsquigarrow} A'$ and $A \overset{d}{\rightsquigarrow} A''$, then $A' = A''$.
- *Time additivity*: If $A \overset{d_1}{\rightsquigarrow} A' \overset{d_2}{\rightsquigarrow} A''$ then $A \overset{d_1+d_2}{\rightsquigarrow} A''$.
- *Time continuity*: If $A \overset{d_1+d_2}{\rightsquigarrow} A''$ then there exists A' such that $A \overset{d_1}{\rightsquigarrow} A' \overset{d_2}{\rightsquigarrow} A''$.

We can now define how a system as a whole (a composition of agents) evolves. There are two kind of transitions: (i) action transitions of the form $\mathcal{A} \xrightarrow{a,S} \mathcal{A}'$ will indicate that system $\mathcal{A}$ can perform action $a \in$ Act with agents indexed by $S \in 2^{\mathcal{I}}$ 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 with the passing of time.

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

- $\mathcal{A} \xrightarrow{a,S} \mathcal{A}'$, with $S \in 2^{\mathcal{I}}$ and $1 \leqslant \#S \leqslant 2$, 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: $\forall i \in \mathcal{I} \cdot i \notin S \Rightarrow A_i = A'_i$; and (iii) agents indexed in S evolve over action a: $\forall i \in \mathcal{I} \cdot i \in S \Rightarrow A_i \xrightarrow{a} A'_i$.

Table 1
A piece of the *Boarding System* contract norms.

Clause	Agent	Modality	Action	Reparation Clause	Time Restriction
0	*Passenger*	*Permission*	Go to the checkin desk (`checkin`)	$\varnothing$	$t_0 - 120$
1	*Passenger*	*Obligation*	Present boarding pass (`PBP`)	$\varnothing$	5
2	*Passenger*	*Obligation*	Show her passport (`ShP`)	$\varnothing$	5
3	*Passenger*	*Permission*	Board (`board`)	8 & 9	90
4	*Passenger*	*Permission*	Board with hand luggage (`hl`)	8 & 9	10
5	*Passenger*	*Prohibition*	Carry in her hand luggage weapons (`weapon`) until landing (`landing`)	$\varnothing$	$t_{landing}$
6	*Passenger*	*Prohibition*	Carry in her hand luggage liquids (`liq`) until boarding (`board`)	8	120
7	*Passenger*	*Obligation*	Dispose of liquids (`dliq`)	$\varnothing$	10
8	*Airline Company*	*Obligation*	Put her hand luggage in the hold (`hlhold`)	$\varnothing$	20
9	*Airline Company*	*Obligation*	Allow passenger to board (`board`)	$\varnothing$	90

- $\mathcal{A} \overset{d}{\rightsquigarrow} \mathcal{A}'$ 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: $\mathcal{A} \xrightarrow{a,s}\!\!\!\!\!\!\rightarrow$.

Proposition 1. Based on this definition and the assumptions we made on the time transitions of agents, we can be shown that systems also obey the following properties:

- *Time determinism*: If $\mathcal{A} \overset{d}{\rightsquigarrow} \mathcal{A}'$ and $\mathcal{A} \overset{d}{\rightsquigarrow} \mathcal{A}''$ then $\mathcal{A}' = \mathcal{A}''$.
- *Time additivity*: If $\mathcal{A} \overset{d_1}{\rightsquigarrow} \mathcal{A}'$ and $\mathcal{A}' \overset{d_2}{\rightsquigarrow} \mathcal{A}''$ then $\mathcal{A} \overset{d_1+d_2}{\rightsquigarrow} \mathcal{A}''$.
- *Time continuity*: If $\mathcal{A} \overset{d_1+d_2}{\rightsquigarrow} \mathcal{A}''$ then there exists $\mathcal{A}'$ such that $\mathcal{A} \overset{d_1}{\rightsquigarrow} \mathcal{A}' \overset{d_2}{\rightsquigarrow} \mathcal{A}''$.

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, $k \in \mathcal{I}$ ranges over agent indices, $a \in$ Act over actions, and $P, Q \in \mathcal{P}$ over predicates.

Predicates tt and ff denote true and false respectively. Predicate $< a, k >$ means that agent k may perform action a. However, since some actions may require involvement by several agents, we use the predicate $< a, \overline{k} >$ 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 $< ticket, c >$ 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 $< ticket, \overline{c} >$ 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:

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

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 number of properties of predicate strength which will be used later on in the paper.

Proposition 2. Let $P, Q \in \mathcal{P}$, k be an agent index and $a \in$ Act, then

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

Proof.

1) Since $P \not\dashv\models tt$ there is a system $\mathcal{A}$ such that $\mathcal{A} \not\models P$. In order to prove $\langle a, k\rangle \vee P \not\dashv\models tt$ we have to find a system $\mathcal{A}'$ such that $\mathcal{A}' \not\models \langle a, k\rangle \vee P$. If $\mathcal{A} \not\models \langle a, k\rangle$ we can take $\mathcal{A}$ as $\mathcal{A}'$ and we are done. So we assume $\mathcal{A} \models \langle a, k\rangle$. System $\mathcal{A}'$ is built by removing all transitions labelled with a from agent k. Clearly $\mathcal{A}' \not\models \langle a, k\rangle$. To show that $\mathcal{A}' \not\models P$ let us proceed by structural induction on P. The only non-trivial case is $P = \langle a, \overline{l}\rangle$. If $l = k$ by definition $\mathcal{A}' \not\models \langle a, \overline{l}\rangle$. If $l \neq k$ and $\mathcal{A}' \models \langle a, \overline{l}\rangle$ then an agent $m \neq k$ that is able to perform action a. Since agents l and k

have not been modified we obtain $\mathcal{A} \models \langle a, \overline{l} \rangle = P$, which is a contradiction.
Now let us prove $\langle a, \overline{k} \rangle \vee P \not\rightrightarrows\!\!\!\not\models tt$. Again, since $P \not\rightrightarrows\!\!\!\not\models tt$ there is a system $\mathcal{A}$ such that $\mathcal{A} \nVdash P$. If $\mathcal{A} \nVdash \langle a, \overline{k} \rangle$ we are already done. So let us assume $\mathcal{A} \models \langle a, \overline{k} \rangle$. In this case we obtain a new system $\mathcal{A}'$ by replacing action a by a new action new (not appearing in $\mathcal{A}$ or in P). It is clear that $\mathcal{A}' \nVdash \langle a, \overline{k} \rangle$. The proof of $\mathcal{A}' \nVdash P$ uses induction on P. All cases are quite simple taking into account that new actions cannot appear in P.

2) The proof is by structural induction on P:
   - *Base cases*: If $P = ff$ then $P \wedge Q \not\rightrightarrows\!\!\!\not\models tt$. If $P = tt$ then the result is immediate. The cases $P = \langle a, k \rangle$ and $P = \langle a, \overline{k} \rangle$ are a consequence of the previous result: if $Q \not\rightrightarrows\!\!\!\not\models tt$ then $\langle a, k \rangle \vee Q \not\rightrightarrows\!\!\!\not\models tt$ and $\langle a, \overline{k} \rangle \vee Q \not\rightrightarrows\!\!\!\not\models tt$.
   - *Inductive cases*: If $Q \rightrightarrows\!\!\models tt$ there is nothing to prove, so let us assume $Q \not\rightrightarrows\!\!\!\not\models tt$.
     - $P = P_1 \wedge P_2$. First let us check that any system $\mathcal{A}$ verify $\mathcal{A} \models P_1 \vee Q$ and $\mathcal{A} \models P_2 \vee Q$, that implies $P_1 \vee Q \rightrightarrows\!\!\models tt$ and $P_2 \vee Q \rightrightarrows\!\!\models tt$. Since $P \vee Q \rightrightarrows\!\!\models tt$ we obtain $\mathcal{A} \models P \vee Q$. If $\mathcal{A} \models Q$ then $\mathcal{A} \models P_1 \vee Q$ and $\mathcal{A} \models P_2 \vee Q$. Otherwise $\mathcal{A} \models P$, so $\mathcal{A} \models P_1$ and $\mathcal{A} \models P_2$ and therefore $\mathcal{A} \models P_1 \vee Q$ and $\mathcal{A} \models P_2 \vee Q$. Since $P_1 \vee Q \rightrightarrows\!\!\models tt$, we can apply the inductive hypothesis to obtain $P_1 \rightrightarrows\!\!\models tt$ or $Q \rightrightarrows\!\!\models tt$. The latter is impossible, so $P_1 \rightrightarrows\!\!\models tt$. In a symmetric way, we obtain $P_2 \rightrightarrows\!\!\models tt$. Now since $P_1 \rightrightarrows\!\!\models tt$ and $P_2 \rightrightarrows\!\!\models tt$, we obtain $P = P_1 \wedge P_2 \rightrightarrows\!\!\models tt$.
     - $P = P_1 \vee P_2$. It is easy to check that the $\vee$ is associative so we obtain $P_1 \vee (P_2 \vee Q) \rightrightarrows\!\!\models tt$. We can apply induction hypothesis to P_1 to obtain $P_1 \rightrightarrows\!\!\models tt$ or $P_2 \vee Q \rightrightarrows\!\!\models tt$. Let us assume the first case, since $P_1 \rightrightarrows\!\!\models tt$, it is easy to check that $P = P_1 \vee P_2 \rightrightarrows\!\!\models tt$. In the second case $P_2 \vee Q \rightrightarrows\!\!\models tt$, we can apply induction hypothesis to P_2 to obtain $P_2 \rightrightarrows\!\!\models tt$ or $Q \rightrightarrows\!\!\models tt$. The second case is impossible and, as before, in the first case we obtain $P = P_1 \vee P_2 \rightrightarrows\!\!\models tt$.

□

5 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.

5.1 Contract Syntax

Definition 5. The set of formulae φ in our contract calculus follows this syntax:

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

where $a \in \mathsf{Act}$, $x \in \mathsf{fvars}$, $k \in \mathcal{I}$ and $d \in \mathbb{T} \cup \{\infty\}$. The set of contract formulae is denoted by C.

The basic formulae $\top$ and $\bot$ indicate, respectively, the contracts which 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 $\mathcal{O}_k(a)[d]$ 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: $\mathsf{wait}(2); \mathcal{O}_k(a)[3]$.

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., $\mathcal{O}_1(a)[2] \blacktriangleright \mathcal{P}_2(b)[5]$ 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 $\mathsf{cond}_k(a)[d](\varphi_1, \varphi_2)$ is a conditional contract which (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 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., $\mathsf{rec}\ x.\mathcal{O}_p(a)[d]; x$ is the contract which obliges agent p to repeatedly perform action a within d time units of each other. In contrast, $\mathsf{rec}\ x.\mathcal{O}_r(pr)[10] \wedge \mathsf{wait}(30); x$, is the contract in which agent r is obliged to pay the rent (action pr) during the first 10 days of every month, repeatedly.

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:

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

Example 1. The contract of the plane boarding system from Section 3, can be formalised using our contract calculus as follows:

$$\begin{array}{rcl} \varphi_0 & ::= & \mathcal{P}_p(\texttt{checkin})[t_0 - 120] \\ \varphi_1 & ::= & \mathcal{O}_p(\texttt{PBP})[5] \\ \varphi_2 & ::= & \mathcal{O}_p(\texttt{ShP})[5] \\ \varphi_3 & ::= & (\mathcal{F}([\texttt{weapon}, p]\,\mathcal{U}\,[\texttt{landing}, c])) \wedge \\ & & ((\mathcal{F}([\texttt{liq}, p]\,\mathcal{U}\,[\texttt{boarding}, p])) \blacktriangleright \mathcal{O}_p(\texttt{dliq})[10]) \\ \varphi_4 & ::= & (\mathcal{P}_p(\texttt{board})[90]; \mathcal{P}_p(\texttt{hl})[10]) \blacktriangleright \\ & & (\mathcal{O}_c(\texttt{board})[90];\ \mathcal{O}_c(\texttt{hlhold})[20]) \end{array}$$

$$PBS \quad ::= \quad \varphi_0; \varphi_1; \varphi_2; (\varphi_3 \wedge \varphi_4)$$

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 this 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*.

We define the notion of closed formulae as the property that the contract formula contains no free recursion variables (a recursion variable x is free if it not bound to a $\mathsf{rec}\ x$ above it).

Definition 6. Let φ be a formula. We define the set of free variables of φ, written $\mathsf{fv}(\varphi)$, as:

$$\begin{aligned}
\mathsf{fv}(\top) &\stackrel{\text{df}}{=} \varnothing & \mathsf{fv}(\varphi \wedge \varphi') &\stackrel{\text{df}}{=} \mathsf{fv}(\varphi) \cup \mathsf{fv}(\varphi') \\
\mathsf{fv}(\bot) &\stackrel{\text{df}}{=} \varnothing & \mathsf{fv}(\varphi \vee \varphi') &\stackrel{\text{df}}{=} \mathsf{fv}(\varphi) \cup \mathsf{fv}(\varphi') \\
\mathsf{fv}(\mathcal{P}_k(a)[d]) &\stackrel{\text{df}}{=} \varnothing & \mathsf{fv}(\varphi \blacktriangleright \varphi') &\stackrel{\text{df}}{=} \mathsf{fv}(\varphi) \cup \mathsf{fv}(\varphi') \\
\mathsf{fv}(\mathcal{O}_k(a)[d]) &\stackrel{\text{df}}{=} \varnothing & \mathsf{fv}(\mathsf{cond}_k(a)[d](\varphi, \varphi')) &\stackrel{\text{df}}{=} \mathsf{fv}(\varphi) \cup \mathsf{fv}(\varphi') \\
\mathsf{fv}(\mathcal{F}_k(a)[d]) &\stackrel{\text{df}}{=} \varnothing & \mathsf{fv}(\varphi; \varphi') &\stackrel{\text{df}}{=} \mathsf{fv}(\varphi) \cup \mathsf{fv}(\varphi') \\
\mathsf{fv}(\mathsf{wait}(d)) &\stackrel{\text{df}}{=} \varnothing & \mathsf{fv}(\mathsf{rec}\ x.\varphi) &\stackrel{\text{df}}{=} \mathsf{fv}(\varphi) \backslash \{x\} \\
\mathsf{fv}(x) &\stackrel{\text{df}}{=} \{x\}
\end{aligned}$$

We say that a formula φ is *closed* iff $\mathsf{fv}(\varphi) = \varnothing$.

A strong prefixed formula is one where all the occurrences of the formula variables are prefixed by an obligation, prohibition, permission or wait operation.

Definition 7. We define the *strong prefixed* predicate over formulas, $\mathsf{sp} : \mathcal{C} \mapsto \mathsf{Bool}$, as follows:

$$\begin{aligned}
\mathsf{sp}(\top) &\stackrel{\text{df}}{=} \mathit{true} & \mathsf{sp}(\varphi; \varphi') &\stackrel{\text{df}}{=} \mathsf{sp}(\varphi) \\
\mathsf{sp}(\bot) &\stackrel{\text{df}}{=} \mathit{true} & \mathsf{sp}(\varphi \wedge \varphi') &\stackrel{\text{df}}{=} \mathsf{sp}(\varphi) \wedge \mathsf{sp}(\varphi') \\
\mathsf{sp}(\mathcal{P}_k(a)[d]) &\stackrel{\text{df}}{=} d > 0 & \mathsf{sp}(\varphi \vee \varphi') &\stackrel{\text{df}}{=} \mathsf{sp}(\varphi) \wedge \mathsf{sp}(\varphi') \\
\mathsf{sp}(\mathcal{O}_k(a)[d]) &\stackrel{\text{df}}{=} d > 0 & \mathsf{sp}(\varphi \blacktriangleright \varphi') &\stackrel{\text{df}}{=} \mathsf{sp}(\varphi) \wedge \mathsf{sp}(\varphi') \\
\mathsf{sp}(\mathcal{F}_k(a)[d]) &\stackrel{\text{df}}{=} d > 0 & \mathsf{sp}(\mathsf{cond}_k(a)[d](\varphi, \varphi')) &\stackrel{\text{df}}{=} \mathsf{sp}(\varphi) \wedge \mathsf{sp}(\varphi') \\
\mathsf{sp}(\mathsf{wait}(d)) &\stackrel{\text{df}}{=} d > 0 & \mathsf{sp}(\mathsf{rec}\ x.\varphi) &\stackrel{\text{df}}{=} \mathsf{sp}(\varphi) \\
\mathsf{sp}(x) &\stackrel{\text{df}}{=} \mathit{false}
\end{aligned}$$

We say that a formula φ is *strong prefixed* iff $\mathsf{sp}(\varphi)$ holds.

5.2 Syntactical Congruence

In order to simplify the presentation of the operational semantics, we will follow an approach similar to that used in the π-calculus [39]. As in other such approaches, we start by defining a given (assumed) 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 8. 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. $\top \blacktriangleright \varphi \equiv \top$
12. $\bot \blacktriangleright \varphi \equiv \varphi$
13. $\mathcal{O}_k(a)[0] \equiv \bot$
14. $\mathcal{F}_k(a)[0] \equiv \top$
15. $\mathcal{P}_k(a)[0] \equiv \top$
16. $\mathsf{wait}(0) \equiv \top$
17. $\mathsf{cond}_k(a)[0](\varphi, \psi) \equiv \psi$

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 ($\mathcal{O}_k(a)[0] \equiv \bot$) allows us to rewrite $\mathcal{O}_k(a)[0]; \mathcal{P}_l(b)[5]$ to $\bot; \mathcal{P}_l(b)[5]$, which in turn can be rewritten to $\bot$ using rule 10 ($\bot; \varphi \equiv \bot$).

Definition 9. 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: $\mathcal{O}_p(\mathtt{PBP})[5]$. In this case, equivalence rule 13 can be applied as soon as 5 time units have passed: $\mathcal{O}_p(\mathtt{PBP})[5] \overset{5}{\rightsquigarrow} \mathcal{O}_p(\mathtt{PBP})[0] \hookrightarrow \bot$.

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 3. The $\hookrightarrow \in \mathcal{C} \leftrightarrow \mathcal{C}$ relation is: (i) *terminating*: there is no infinite sequence φ_1, φ_2 ..., 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 [40].

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 10. A contract formule $\varphi \in \mathcal{C}$ 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 $\varphi \longmapsto \varphi'$.

5.3 Operational Semantics

We can now define an operational semantics for our contract calculus. 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.

Table 2
Obligation transition rules.

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

Table 3
Prohibition transition rules.

(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)	$\mathcal{F}_k(a)[d] \overset{d'}{\rightsquigarrow} \mathcal{F}_k(a)[d-d'], \quad 0 < d' \leqslant d$

Rules **O1**, **O2**, **O3**, **O4**, and **O5** (Table 2) define the behaviour of obligations $\mathcal{O}_k(a)[d]$, i.e., the obligation on agent k to perform action a within d time units. 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 with the syntactic equivalence $\mathcal{O}_k(a)[0] \equiv \bot$, it is not necessary to explicitly handle this case in the semantics.

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

Rules **F1**, **F2**, **F3**, **F4**, and **F5** (Table 3) define the cases for prohibition similar to obligation.

Table 4
Permission transition rules.

(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)	$\mathcal{P}_k(a)[d] \overset{d'}{\rightsquigarrow} \mathcal{P}_k(a)[d-d']\,, \quad 0 < d' \leqslant d$

Table 5
Wait transition rule.

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

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** (Table 4). 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 involved 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[1]. Rule **P4** considers the case when other agent than k intents 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.

1. It is worth noting that this interpretation of permission makes sense in an interactive systems setting, where permission to perform an action corresponds to the other parties being willing to allow one to proceed with that action.

The wait rules are presented in Table 5, which 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.

Table 6
Condition transition rules.

(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)	$\mathrm{cond}_k(a)[d](\varphi,\psi) \overset{d'}{\rightsquigarrow} \mathrm{cond}_k(a)[d-d'](\varphi,\psi), \quad 0 < d' \leqslant d$

The rules for conditional contracts (Table 6) 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.

Table 7
Transition rules for conjunction and disjunction ($\mathsf{op} \in \{\vee, \wedge\}$)

(AO1)	$\dfrac{\varphi \xrightarrow{\alpha} \varphi',\ \psi \xrightarrow{\alpha} \psi'}{\varphi \ \mathsf{op}\ \psi \xrightarrow{\alpha} \varphi' \ \mathsf{op}\ \psi'}$
(AO2)	$\dfrac{\varphi \overset{d}{\rightsquigarrow} \varphi',\ \psi \overset{d}{\rightsquigarrow} \psi'}{\varphi \ \mathsf{op}\ \psi \overset{d}{\rightsquigarrow} \varphi' \ \mathsf{op}\ \psi'}$
(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)	$\dfrac{\varphi \overset{d}{\rightsquigarrow} \varphi',\ \psi \overset{d'}{\rightsquigarrow} \bot}{\varphi \vee \psi \overset{d}{\rightsquigarrow} \varphi'},\ d \geqslant d'$

The rules for conjunction and disjunction (Table 7) 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 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 have been already violated and how the disjunction of both contracts evolve, in an analogous manner as conjunction.

Table 8
Recovery and sequential transition rules

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

The rules for reparation and sequential composition are presented in Table 8. The first two 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 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: $\varphi_4 ::= (\mathcal{P}_p(\texttt{board})[90]; \mathcal{P}_p(\texttt{hl})[10]) \blacktriangleright (\mathcal{O}_c(\texttt{board})[90];\ \mathcal{O}_c(\texttt{hlhold})[20])$, where the passenger is permitted to board within 90 minutes ($\mathcal{P}_p(\texttt{board})[90]$) 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 ($\mathcal{O}_c(\texttt{board})[90]$) and then, to put the passenger's hand luggage in the hold within 20 minutes ($\mathcal{O}_c(\texttt{hlhold})[20]$). If 90 time units pass, φ_4 evolves in the following way:

$\mathcal{P}_p(\texttt{board})[90]; \mathcal{P}_p(\texttt{hl})[10] \blacktriangleright \mathcal{O}_c(\texttt{board})[90];\ \mathcal{O}_c(\texttt{hlhold})[20]$
$\overset{90}{\rightsquigarrow}$
$\mathcal{P}_p(\texttt{board})[0]; \mathcal{P}_p(\texttt{hl})[10] \blacktriangleright (\mathcal{O}_c(\texttt{board})[90];\ \mathcal{O}_c(\texttt{hlhold})[20]$

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

Table 9
Recursion transitions

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

The final rules (Table 9) 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 expression 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 4. 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 $k \in \mathcal{I}$, $\varphi \xrightarrow{a,k}$ 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 5. 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:

- *Time determinism*: If $\varphi \overset{d_1}{\rightsquigarrow} \varphi'$ and $\varphi \overset{d_1}{\rightsquigarrow} \varphi''$ then $\varphi' \equiv \varphi''$.
- *Time additivity*: If $\varphi \overset{d_1}{\rightsquigarrow} \varphi' \overset{d_2}{\rightsquigarrow} \varphi''$, then $\varphi \overset{d_1+d_2}{\rightsquigarrow} \varphi''$.
- *Time continuity*: If $\varphi \overset{d_1+d_2}{\rightsquigarrow} \varphi''$ then there exists $\varphi' \in \mathcal{C}$ 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 $\mathcal{O}_k(a)[d]$, $\mathcal{F}_k(a)[d]$, $\mathcal{P}_k(a)[d]$ 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). □

5.4 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 11. 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:

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

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 a 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, violation can only occur after the time has expired ($d = 0$), but this case is already defined because of the syntactical equivalence $\mathcal{O}_k(a)[0] \equiv \bot$. 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 $\mathsf{vio}(\varphi \triangleright \varphi')$, 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.

Another important property that will be used later is established in the following lemma.

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 2. □

Note that the analogous property for $\top$ does not hold, i.e., there are contracts for which $\mathsf{vio}(\varphi) \dashv\models \mathit{ff}$, but $\varphi \not\equiv \top$.

5.5 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 12. Given a contract $\varphi \in \mathcal{C}$ with alphabet 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)	$\dfrac{\varphi \xrightarrow{\overline{(a,k)}} \varphi',\ \mathcal{A} \vDash \langle a, \overline{k} \rangle}{\varphi \parallel \mathcal{A} \Rightarrow \varphi' \parallel \mathcal{A}}$
(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 which the contract is not interested in. Finally, rule **M4** ensures that time cannot skip over a violation.

Definition 13. Let $\mathcal{A}$ be a system and $\varphi \in \mathcal{C}$ be a contract.

- System $\mathcal{A}$ can *break* φ, written $\mathsf{break}(\mathcal{A}, \varphi)$, 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:

  $$\varphi \| \mathcal{A} = \varphi_0 \| \mathcal{A}_0 \Rrightarrow \varphi_1 \| \mathcal{A}_1 \Rrightarrow \ldots \varphi_{n-1} \| \mathcal{A}_{n-1} \Rrightarrow \varphi_n \| \mathcal{A}_n,$$

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

  $$\varphi \| \mathcal{A} = \varphi_0 \| \mathcal{A}_0 \Rrightarrow \varphi_1 \| \mathcal{A}_1 \Rrightarrow \ldots \varphi_{n-1} \| \mathcal{A}_{n-1} \Rrightarrow \varphi_n \| \mathcal{A}_n\ ,$$

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

Note that there are contracts which 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.

6 Refinement

In this section we define two notions of contract refinement, one ($\preccurlyeq_\bot$) related to the violation of a contract $\bot$, and another ($\preccurlyeq_\top$) related to the fulfilment of a contract $\top$.

Intuitively $\preccurlyeq_\bot$ relates two contracts $\varphi, \psi \in \mathcal{C}$ (i.e. $\varphi \preccurlyeq_\bot \psi$) if any system which can break contract φ, can also break 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. We start by defining what a $\top$ (respectively $\bot$) simulation contract is (Definition 14 and respectively Definition 15), using which we can then define the $\top$ (respectively $\bot$) simulation as the union of all $\top$ (respectively $\bot$) simulations (Definition 16).

Definition 14. Let $\varphi, \psi \in \mathcal{C}$ and $R \subseteq \mathcal{C} \times \mathcal{C}$, we say that R is a $\bot$ *simulation contract* relation iff whenever $(\varphi, \psi) \in R$ the following conditions hold:

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

(ii) If $\varphi \overset{d}{\rightsquigarrow} \varphi'$ then one of the following conditions holds:
   a. there exists $d' \leqslant d$ such that $\psi \overset{d'}{\rightsquigarrow} \bot$, or
   b. there exists $\psi' \in \mathcal{C}$ 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 $(\varphi', \psi') \in R$.

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

Proof. It is immediate from the definitions. □

Lemma 3. Let R_1 and R_2 be $\bot$ simulation contract relations. Then, their composition $R_1 \circ R_2$ is also a $\bot$ simulation contract relation.

Proof. Let us consider $(\varphi_1, \varphi_2) \in R_1 \circ R_2$. By definition, there exists $\psi \in \mathcal{C}$ such that $(\varphi_1, \psi) \in R_1$ and $(\psi, \varphi_2) \in R_2$. Let us check that (φ_1, φ_2) satisfies the conditions of Definition 14:

(i) Since R_1 is a $\bot$ simulation relation, $\mathsf{vio}(\varphi_1) \vDash \mathsf{vio}(\psi)$. Since R_2 is a $\bot$ simulation contract relation, $\mathsf{vio}(\psi) \vDash \mathsf{vio}(\varphi_2)$. Let us consider a system $\mathcal{A}$ such that $\mathcal{A} \vDash \mathsf{vio}(\varphi_1)$. Since $\mathsf{vio}(\varphi_1) \vDash \mathsf{vio}(\psi)$, we obtain $\mathcal{A} \vDash \mathsf{vio}(\psi)$. Since $\mathsf{vio}(\psi) \vDash \mathsf{vio}(\varphi_2)$, we obtain $\mathcal{A} \vDash \mathsf{vio}(\varphi_2)$.

(ii) Let us assume $\varphi_1 \overset{d}{\rightsquigarrow} \varphi_1'$. Since R_1 is a $\bot$ simulation contract relation, there are two cases:
   a. There exists $d' \leqslant d$ such that $\psi \overset{d'}{\rightsquigarrow} \bot$. Since R_2 is a *bot* simulation contract relation, there are two cases:
      1. There exists $d'' \leqslant d'$ such that $\varphi_2 \overset{d''}{\rightsquigarrow} \bot$. In this case there is nothing left to prove.
      2. There exists $\varphi_2' \in \mathcal{C}$ such that $\varphi_2 \overset{d'}{\rightsquigarrow} \varphi_2'$ and $(\bot, \varphi_2') \in R_2$. Since R_2 is a $\bot$ simulation contract relation and $\mathsf{vio}(\bot) = tt$, then $\mathsf{vio}(\varphi_2') = tt$. By Lemma 1, we obtain $\varphi_2' = \bot$.
   b. There is ψ' such that $\psi \overset{d}{\rightsquigarrow} \psi'$ and $(\varphi_1', \psi') \in R_1$. Since R_2 is a $\bot$ simulation contract relation, there are two cases:
      1. There exists $d' \leqslant d$ such that $\varphi_2 \overset{d'}{\rightsquigarrow} \bot$. In this case there is no more to prove.
      2. There exists $\varphi_2' \in \mathcal{C}$ such that $\varphi_2 \overset{d}{\rightsquigarrow} \varphi_2'$ and $(\psi', \varphi_2') \in R_2 \in R_2$. Since $(\varphi_1', \psi') \in R_1$ and $(\psi', \varphi_2') \in R_2$, we obtain $(\varphi_1', \varphi_2') \in R_1 \circ R_2$.

(iii) Let us assume $\varphi_1 \xrightarrow{a} \varphi_1'$. Since R_1 is a $\bot$ simulation contract relation, there is $\psi \in \mathcal{C}$ such that $\psi \xrightarrow{a} \varphi$ and $(\varphi_1', \psi) \in R_1$. Since R_2 is a $\bot$ simulation contract relation, there is $\varphi_2' \in \mathcal{C}$ such that $\varphi_2 \xrightarrow{a} \varphi_2'$ and $(\psi, \varphi_2') \in R_2$. Since, $(\varphi_1', \psi) \in R_1$ and $(\psi, \varphi_2') \in R_2$ we obtain by definition $(\varphi_1', \varphi_2') \in R_1 \circ R_2$ □

Proposition 6. The relation $\preccurlyeq_\perp$ is reflexive and transitive.

Proof.

- *Reflexivity*. Since $id = \{(\varphi, \varphi) \mid \varphi \in \mathcal{C}\}$ is a $\perp$ simulation contract relation, we obtain $\varphi \preccurlyeq_\perp \varphi$ for any $\varphi \in \mathcal{C}$.
- *Transitivity*. Let $\varphi_1, \varphi_2, \varphi_3 \in \mathcal{C}$ such that $\varphi_1 \preccurlyeq_\perp \varphi_2$ and $\varphi_2 \preccurlyeq_\perp \varphi_3$. That means that there are two $\perp$ simulation contract relations R_1 and R_2 such that $(\varphi_1, \varphi_2) \in R_1$ and $(\varphi_2, \varphi_3) \in R_2$. Therefore, $(\varphi_1, \varphi_3) \in R_1 \circ R_2$. Since $R_1 \circ R_2$ is a $\perp$ simulation contract relation, we obtain $\varphi_1 \preccurlyeq_\perp \varphi_3$.

□

Definition 15. Let $\varphi, \psi \in \mathcal{C}$ and $R \subseteq \mathcal{C} \times \mathcal{C}$, we say that R is a $\top$ *simulation contract* 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 one of the following conditions hold:
  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 $(\varphi', \psi') \in R$

Lemma 4. The relation $id = \{(\varphi, \varphi) \mid \varphi \in \mathcal{C}\}$ is a $\top$ simulation relation.

Proof. It is immediate from the definitions. □

Lemma 5. Let R_1 and R_2 be $\top$ simulation contract relations. Then, their composition $R_1 \circ R_2$ is also a $\top$ simulation contract relation.

Proof. Let us consider $(\varphi_1, \varphi_2) \in R_1 \circ R_2$. By definition, there exists $\psi \in \mathcal{C}$ such that $(\varphi_1, \psi) \in R_1$ and $(\psi, \varphi_2) \in R_2$. Let us check that (φ_1, φ_2) satisfies the conditions of Definition 15:

(i) Let us assume $\varphi_1 \equiv \top$. Since R_1 is a $\top$ simulation contract relation, $\psi \equiv \top$. Since R_2 is a $\top$ simulation contract relation, $\varphi_2 \equiv \top$
(ii) Since R_2 is a $\top$ simulation contract relation, $\mathsf{vio}(\varphi_2) \models \mathsf{vio}(\psi)$. Since R_1 is a $\top$ simulation contract relation, $\mathsf{vio}(\psi) \models \mathsf{vio}(\varphi_1)$. Therefore, $\mathsf{vio}(\varphi_2) \models \mathsf{vio}(\varphi_1)$.
(iii) Let us assume $\varphi_1 \overset{d}{\rightsquigarrow} \varphi_1'$. Since R_1 is a $\perp$ simulation contract relation, there are two cases:
  a. There exists $d' \leqslant d$ such that $\psi \overset{d'}{\rightsquigarrow} \perp$. Since R_2 is a *bot* simulation contract relation there are two cases:
    1. There exists $d'' \leqslant d'$ such that $\varphi_2 \overset{d''}{\rightsquigarrow} \perp$. In this case there is no more to prove.
    2. There exists $\varphi_2' \in \mathcal{C}$ such that $\varphi_2 \overset{d'}{\rightsquigarrow} \varphi_2'$ and $(\perp, \varphi_2') \in R_2$. Since R_2 is a $\perp$ simulation contract relation, we obtain $\varphi_2' = \perp$.
  b. There exists ψ' such that $\psi \overset{d}{\rightsquigarrow} \psi'$ and $(\varphi_1', \psi') \in R_1$. Since R_2 is a $\perp$ simulation contract relation, there are two cases:
    1. There exists $d' \leqslant d$ such that $\varphi_2 \overset{d'}{\rightsquigarrow} \perp$. In this case there is nothing more to prove.
    2. There exists $\varphi_2' \in \mathcal{C}$ such that $\varphi_2 \overset{d}{\rightsquigarrow} \varphi_2'$ and $(\psi', \varphi_2') \in R_2$. Since $(\varphi_1', \psi') \in R_1$ and $(\psi', \varphi_2') \in R_2$, we obtain $(\varphi_1', \varphi_2') \in R_1 \circ R_2$.
(iv) Let us assume $\varphi_1 \xrightarrow{a} \varphi_1'$. Since R_1 is a $\perp$ simulation contract relation, there exists $\psi \in \mathcal{C}$ such that $\psi \xrightarrow{a} \varphi$ and $(\varphi_1', \psi) \in R_1$. Since R_2 is a $\perp$ simulation contract relation, there exists $\varphi_2' \in \mathcal{C}$ such that $\varphi_2 \xrightarrow{a} \varphi_2'$ and $(\psi, \varphi_2') \in R_2$. Since, $(\varphi_1', \psi) \in R_1$ and $(\psi, \varphi_2') \in R_2$ we obtain by definition $(\varphi_1', \varphi_2') \in R_1 \circ R_2$

□

Proposition 7. The relation $\preccurlyeq_\top$ is reflexive and transitive.

Proof. This proof is identical to that of Proposition 6, replacing $\perp$ by $\top$. □

We can now define the notion of simulation.

Definition 16. A contract φ *can be* $\top$ *simulated (respectively* $\perp$ *simulated)* by the contract ψ (written $\varphi \preccurlyeq_\top \psi$, respectively written $\varphi \preccurlyeq_\perp \psi$) iff there exists a $\top$ simulation contract relation (respectively $\perp$ simulation contract relation) R such that $(\varphi, \psi) \in R$.

Consider the following example illustrating the use of these definitions.

Example 5.

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

Starting from the inequalities given in the the left-hand column, it is not difficult to formally verify their correctness. For instance, consider $\mathsf{wait}(3)$ and $\mathcal{P}_k(a)[5]$ — the first one cannot be violated while the second can be violated by any system that does not allow agent k to perform action a, which ensures that the simulation holds. Similar reasoning can be used to show the relation between contracts $\mathsf{wait}(5)$ and $\mathcal{O}_k(a)[6]$. Regarding the $\preccurlyeq_\top$ relation, dual reasoning can be applied — $\mathcal{P}_k(a)[3]$ can only be fulfilled if agent k is allowed to perform a within 3 units, whereas $\mathsf{wait}(3)$ cannot be violated.
As to the relations shown in the right-hand column, starting from $\mathsf{wait}(3) \preccurlyeq_\perp \mathcal{P}_k(a)[5]$, we can put both contracts in the context of the continuation operator to follow up with $\perp$. While $\mathsf{wait}(3); \perp$ cannot be fulfilled after 3 units of time whatever the system does, in the case of $\mathcal{P}_k(a)[5]; \perp$, 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. 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 8. The two equivalence relations $\backsim_\top = \preccurlyeq_\top \cap \preccurlyeq_\top{}^{-1}$ and $\backsim_\perp = \preccurlyeq_\perp \cap \preccurlyeq_\perp{}^{-1}$, are equal to each other: $\backsim_\top = \backsim_\perp$.

Proof. In order to prove $\backsim_\perp \subseteq \backsim_\top$ we have to prove $\backsim_\perp \subseteq \preccurlyeq_\perp$ and $\backsim_\perp \subseteq \preccurlyeq_\perp{}^{-1}$. Both proofs are symmetrical, so let us prove the first. It is sufficient to prove that $\backsim_\perp$ is a $\top$ simulation relation. Consider $\varphi, \psi \in \mathcal{C}$ such that $\varphi \backsim_\perp \psi$

— we must prove the conditions of Definition 15, with the only non-trivial one being condition i. Assume $\varphi \equiv \top$. Since $\varphi \backsim_\perp \psi$, we deduce $\mathsf{vio}(\psi) = \mathit{ff}$. If $\psi \not\equiv \top$, then by Proposition 4, for any possible α there must exist ψ' such that $\psi \xrightarrow{\alpha} \psi'$. Again, since $\varphi \backsim_\perp \psi$ 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 17. We define the *simulation equivalence relation* as

$$\backsim \stackrel{\text{df}}{=} \preccurlyeq_\top \cap \preccurlyeq_\top^{-1}$$

Consider the $\perp$ simulation: if two contracts are related $\varphi \preccurlyeq_\perp \psi$, 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 $\varphi \preccurlyeq_\perp \psi$. Then, if $\mathcal{A}$ violates φ, it also violates ψ: $\mathsf{break}(\mathcal{A}, \varphi) \Rightarrow \mathsf{break}(\mathcal{A}, \psi)$.

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

$$\varphi \parallel \mathcal{A} = \varphi_0 \parallel \mathcal{A}_0 \Rightarrow \varphi_1 \parallel \mathcal{A}_1 \Rightarrow \ldots \varphi_{n-1} \parallel \mathcal{A}_{n-1} \Rightarrow \varphi_n \parallel \mathcal{A}_n = \varphi' \parallel \mathcal{A}'$$

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

$$\psi \parallel \mathcal{A} = \psi_0 \parallel \mathcal{A}_0 \Rightarrow \psi_1 \parallel \mathcal{A}_1 \Rightarrow \ldots \psi_{m-1} \parallel \mathcal{A}_{m-1} \Rightarrow \psi_m \parallel \mathcal{A}_m = \psi' \parallel \mathcal{A}'$$

such that $m \leqslant n$, $(\varphi_k, \psi_k) \in R$ for $0 \leqslant k < m$, and $\mathsf{break}(\mathcal{A}', \psi_m)$. 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 $\psi_0 \overset{d}{\rightsquigarrow} \perp$. In this case we obtain $\psi \parallel \mathcal{A} \Rightarrow \perp \parallel \mathcal{A}'$.
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 $\psi \parallel \mathcal{A} \Rightarrow^* \perp \parallel \mathcal{A}'$. 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. In this case we build a computation:

$$\psi \parallel \mathcal{A} = \psi_0 \parallel \mathcal{A}_0 \Rightarrow \psi_1 \parallel \mathcal{A}_1 \Rightarrow \ldots \psi_{m-1} \parallel \mathcal{A}_{m-1} \Rightarrow \psi_m \parallel \mathcal{A}_m = \psi' \parallel \mathcal{A}'$$

such that $m \leqslant n$, $(\varphi_k, \psi_k) \in R$, $\mathcal{A}_k \not\models \mathsf{vio}(\psi_k)$ for $0 \leqslant k < m$, and $\psi_m \equiv \top$. The proof is equally done by induction on n. The inductive case ($n = 0$) is also trivial and the recursive cases for rules **M1**, **M2** and **M3** are just very similar. We only have to verify $\mathcal{A}_k \not\models \mathsf{vio}(\psi_k)$, which is immediate since $\mathsf{vio}(\psi_k) \models \mathsf{vio}(\varphi_k)$. The case **M3** 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 5 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 $\preccurlyeq_\perp$. First, let us show that $\top$ and $\perp$ are *the best* contracts in their respective relations $\preccurlyeq_\top$ and $\preccurlyeq_\perp$. Then, as $\varphi \wedge \top \equiv \varphi$ and $\varphi \vee \perp \equiv \varphi$, it is important to show $\varphi \wedge \varphi' \preccurlyeq_\top \varphi$ and $\varphi \vee \varphi' \preccurlyeq_\perp \varphi$.

Proposition 9. For any $\varphi, \varphi' \in \mathcal{C}$, the following hold:

1) $\varphi \preccurlyeq_\top \top$
2) $\varphi \preccurlyeq_\perp \perp$
3) $\varphi \vee \varphi' \preccurlyeq_\perp \varphi$
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 $R_\perp = \{(\varphi \vee \varphi', \varphi) \mid \varphi, \varphi' \in \mathcal{C}\}$ is a $\perp$ 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 $\perp$ simulation contracts. □

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

Proposition 10. For any $\varphi, \varphi', \psi, \psi' \in \mathcal{C}$, the following hold:

If $\varphi' \preccurlyeq_\perp \varphi$ and $\psi' \preccurlyeq_\perp \psi$:	If $\varphi' \preccurlyeq_\top \varphi$ and $\psi' \preccurlyeq_\top \psi$:
$\perp.1$ $\varphi' \triangleright \psi' \preccurlyeq_\perp \varphi \triangleright \psi$	$\top.1$ $\varphi'; \psi' \preccurlyeq_\top \varphi; \psi$
$\perp.2$ $\varphi' \wedge \psi' \preccurlyeq_\perp \varphi \wedge \psi$	$\top.2$ $\varphi' \wedge \psi' \preccurlyeq_\top \varphi \wedge \psi$
$\perp.3$ $\varphi' \vee \psi' \preccurlyeq_\perp \varphi \vee \psi$	$\top.3$ $\varphi' \vee \psi' \preccurlyeq_\top \varphi \vee \psi$
$\perp.4$ $\mathrm{cond}_k(a)[d](\varphi', \psi')$ $\preccurlyeq_\perp \mathrm{cond}_k(a)[d](\varphi, \psi)$	$\top.4$ $\mathrm{cond}_k(a)[d](\varphi', \psi')$ $\preccurlyeq_\top \mathrm{cond}_k(a)[d](\varphi, \psi)$

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 14 and 15. □

A consequence of the previous proposition is that $\backsim$ is a congruence i.e. we can *replace* any subformula of a contract by any other equivalent one without changing the meaning of the contract.

Theorem 3. $\backsim$ is a congruence.

Proof. This follows from the previous proposition and Proposition 8. □

7 Case Study

The case study presented in this section is an extension of the running example described in Section 3, and is based on the Madrid Barajas airport regulations [41], [42], [43]. The following contract between the airline company, the security airport staff, and the passenger regulates their interaction during check-in and on the flight considering time constraints:

1) The passenger is permitted to check in her luggage according to the stipulations of the class of ticket they purchased from their respective airline company. It is necessary that the passenger arrives at the airport, at least, two hours prior to her flight, in order to check-in her luggage and pass the security controls. Then, 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 within 5 minutes.
3) After presenting the boarding pass, the passenger must show her passport, she has 5 minutes for this purpose.
4) The passenger is permitted to carry two pieces (of hand luggage): one personal article and one carry-on luggage. If the passenger has carry-on luggage, she is obliged to fit it into the device for hand luggage allowance, situated next to the check-in desks.
5) After presenting her passport, the passenger is permitted to board within 90 minutes and to present the hand-luggage to the airport staff within 10 minutes.
6) The airline company is obliged to allow the passenger to board within 90 minutes.
7) The passenger is obliged to pass the filters or security checkpoints, before they access the restricted safety areas of the airport, as boarding gates and passenger-only zones, in accordance with the safety regulations, within 60 minutes.
8) These security checkpoints consist of metal-detector arches for the passengers and X-ray detectors for their luggage. The airport security staff are permitted to carry both systems manually.
9) If the hand luggage is a personal computer or another electronic device, the airport security staff is permitted to ask the passenger to take it out of its protective case in order to be examined.
10) The passenger is obliged to take out the personal computer protection if the airport staff need to examine it within 10 minutes.
11) The passenger is forbidden from taking articles to the security restricted area, or to the cabin of the aircraft, which constitute a risk for the health of other passengers, the crew and the safety of the aircraft and the cargo.
12) The passenger is obliged to included risk articles at check-in as baggage and/or apply to them the relevant procedure to be accepted on board. Otherwise, the security staff can requisition the articles.
13) Security staff is permitted to deny access to the boarding area and the airplane cabin to any passenger in possession of an object which, even if not considered forbidden, arouses their suspicions. 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.
14) The passenger is obliged to transport the liquids in individual containers with a capacity of fewer than 100 ml. These containers must be carried in a resealable, transparent plastic bag (for its easy inspection), with a capacity of not more than 1 liter. Maximum one bag per passenger.
15) The passenger is obliged to accompany her medication with a corresponding receipt, a medical prescription or a specified statement about the passenger's health condition, in case the security staff requires it.
    Exception: All liquid medication is excluded from the restrictions, affecting the transport of liquids in hand luggage, if its usage is indispensable for the passenger during the journey and possible extensions: outward flight + stay + return flight.
16) Even if the regulations for liquids do not apply in the medication case, the passenger is obliged to demonstrate all liquid medication to the security staff, apart from the transparent plastic bag, used for the transport of other liquids.
17) The passenger is obliged to check in all firearms, which may not be transported.
18) The passenger has an obligation to know her rights if she wants to file a complaint. In this case, she may ask for a document at any airport in Spain, in which his rights are described, including some advice about how to act. Over and above, the passenger may also contact the Spanish National Aviation Agency (Agencia Estatal de Seguridad Aérea — AESA).
19) The passenger is entitled to present a claim, in case of any violation of those rights, which can result in a financial compensation or any other kind of compensation. If a passenger is unhappy with the service during a flight but is not entitled to present a claim, he still has the option to lodge a complaint or a suggestion.

Table 10 shows the *PBS* contract as a list of the obligations, permissions, and prohibitions that can be inferred

from the description of the process.

The contract describing our case study can be formalized using our contract calculus as follows:

$$\begin{aligned} PBS ::= &((\varphi_0 \wedge \varphi_{10} \wedge \varphi_{15}); \varphi_1; \varphi_2; \varphi_3; \\ &(\varphi_5 \wedge \varphi_6 \wedge \varphi_7 \wedge \varphi_8 \wedge \varphi_9 \wedge \varphi_{12} \wedge \varphi_{13} \wedge \varphi_{14}); \\ &(\varphi_4 \wedge \varphi_{11})) \wedge \varphi_{16} \wedge \varphi_{17}) \end{aligned}$$

Where the formulas $\varphi_1, \ldots, \varphi_{17}$ are defined as follows:

$$\begin{aligned} \varphi_0 ::= &\ \mathcal{P}_p(\texttt{checkin})[t_0 - 120] \\ \varphi_1 ::= &\ \mathcal{O}_p(\texttt{PBP})[5] \\ \varphi_2 ::= &\ \mathcal{O}_p(\texttt{ShP})[5] \\ \varphi_3 ::= &\ \mathcal{P}_p(\texttt{CToHL})[120] \\ \varphi_4 ::= &\ (\mathcal{P}_p(\texttt{board})[90]; \mathcal{P}_p(\texttt{hl})[10]) \\ &\blacktriangleright (\mathcal{O}_c(\texttt{board})[90];\ \mathcal{O}_c(\texttt{hlhold})[20]) \\ \varphi_5 ::= &\ \mathcal{O}_p(\texttt{PSC})[60] \\ \varphi_6 ::= &\ \mathcal{P}_S(\texttt{CD})[120] \\ \varphi_7 ::= &\ \mathcal{P}_S(\texttt{EPP})[120] \\ \varphi_8 ::= &\ \mathcal{O}_p(\texttt{TOP})[120] \\ \varphi_9 ::= &\ (\mathcal{F}([\texttt{TRA}, p]\,\mathcal{U}\,[\texttt{landing}, p])) \\ \varphi_{10} ::= &\ \mathcal{O}_p(\texttt{CRA})[t_0 - 120] \blacktriangleright \mathcal{P}_S(\texttt{RRA})[10] \\ \varphi_{11} ::= &\ \mathcal{P}_S(\texttt{DRA})[10] \\ \varphi_{12} ::= &\ \mathcal{O}_p(\texttt{liquids})[120] \\ \varphi_{13} ::= &\ \mathcal{O}_p(\texttt{medication})[120] \\ \varphi_{14} ::= &\ \mathcal{O}_p(\texttt{liq})[120] \\ \varphi_{15} ::= &\ \mathcal{O}_p(\texttt{firearms})[t_0 - 120] \\ \varphi_{16} ::= &\ \mathcal{O}_p(\texttt{rights})[120] \\ \varphi_{17} ::= &\ \mathcal{O}_p(\texttt{complaint})[120] \end{aligned}$$

Where p, S, c refer to the passenger, the security airport staff, and the airline company, respectively. t_0 is departure estimated time. Note that the clauses φ_0 to φ_{17} are used to express the different parts of the contract, and combined together in the top-level contract expression *PBS*.

Then, this approach allows us to check and determine if any of the agents involved in the plane boarding system break the contract (Definition 13). In this case, our formalism allows us to determine, for instance, if it was the passenger who violated the contract and if so why e.g. because she did not present her boarding pass within the specified time at the check-in desk, or because she was taking articles to the security restricted area.

8 Runtime Verification of Timed Contracts

Although much work has been done on runtime verification [44] — dynamic analysis technique which uses software monitors to identify potential flaws of a system — there is limited work done on runtime verification of real-time properties [45], [46], [47]. Runtime monitors have been used in the literature for a whole spectrum of applications — from simply observing the system under scrutiny but also, beyond this, to verify, enforce properties, or even add functionality to the system.

We will be extending the method to generate a runtime monitor from a contract presented in [11], to handle time.

8.1 Preliminaries

The operational semantics we give to contracts provides us with a framework for contract monitoring: to monitor contract $\psi \in \mathcal{C}$, we start the monitor in state ψ and update the state whenever the system performs an action according to the operational semantics. A violation is reached once the violation predicate is satisfied by the system. In the rest of this section, we concretely show how our logic can be automatically monitored using a derivative-based algorithm [48].

The idea behind derivative-based or term rewriting-based monitoring is that the formula still to be monitored is used as the state of the monitoring system. Whenever an event e is received with the system being in state ψ, the state is updated to ψ' such that any trace of events es matches ψ', if and only if $e : es$ (the trace starting with e, followed by es) matches ψ. This is repeated and a violation is reported when (and if) the monitoring state is reduced to a formula which matches the empty trace. In our contract logic, the operational semantics provide precisely this information, with ψ' being chosen to be the (unique) formula such that[2] $\psi \xrightarrow{a,k}; \longmapsto \psi'$ (where action a performed by party k is observed by the monitoring system), and $\mathsf{vio}(\psi)$ indicates whether ψ matches the empty string (immediately violates the contract).

In timed logics, this approach has to be augmented with timeout events which, in the absence of a system event, still change the formula. For example, the contract $\mathsf{wait}(d); \varphi$ would evolve to φ upon d time units elapsing. Similarly, if d time units elapse (with no system events received), we evolve the contract $\mathcal{O}_k(a)[d]; \varphi$ to $\bot; \varphi$, which is equivalent to $\bot$, thus enabling us to flag the violation as soon as it happens. If we were to wait for a system event, the violation may end up being identified too late. In our case, we use the timeout function to enable setting of a timer to trigger the monitoring state update, evolving φ to the unique formula φ' according to the timed operational semantics $\varphi \overset{\text{timeout}(\varphi)}{\rightsquigarrow}; \longmapsto \varphi'$. Also, system events carry a timestamp, through which the contract can be moved ahead in time upon receiving the event.

In order to formalize these ideas, we need a notion of structural equivalence. Intuitively, two contract are structurally equivalent if they only differ in the time constraints and so they can perform the same actions.

Definition 18. Consider the relation $\mathcal{R} \subseteq \mathcal{C} \times \mathcal{C}$ defined as follows:

$$\begin{aligned} \mathcal{R} \overset{\text{df}}{=} &\ \{(\top, \top), (\bot, \bot)\} \\ \cup &\ \{(\mathcal{F}_k(a)[d], \mathcal{F}_k(a)[d']) \mid d, d' > 0\} \\ \cup &\ \{(\mathcal{P}_k(a)[d], \mathcal{P}_k(a)[d']) \mid d, d' > 0\} \\ \cup &\ \{(\mathcal{O}_a(k)[d], \mathcal{O}_a(k)[d']) \mid d, d' > 0\} \\ \cup &\ \{(\mathsf{wait}(d), \mathsf{wait}(d')) \mid d, d' > 0\} \\ \cup &\ \{(\mathsf{cond}_k(a)[d](\varphi_1, \varphi_2), \mathsf{cond}_k(a)[d'](\varphi_1, \varphi_2)) \mid d, d' > 0, \\ &\quad \varphi_1, \varphi_2 \in \mathcal{C}\} \end{aligned}$$

The *structural equivalence congruence*, denoted by $\equiv_s$ is the smallest congruence containing $\mathcal{R}$.

Proposition 11. Let $\varphi, \psi \in \mathcal{C}$ be two structurally equivalent contracts, $\varphi \equiv_s \psi$. Then $\mathsf{vio}(\varphi) = \mathsf{vio}(\psi)$ and $\varphi \xrightarrow{\alpha} \chi$ iff $\psi \xrightarrow{\alpha} \chi$.

Proof. The proof follows using structural induction. □

Definition 19. The *timeout* of a contract φ, written $\text{timeout}(\varphi)$, is inductively defined as follows:

2. We write $r; s$ to indicate the forward composition of the two relations r and s.

Table 10
Norms of the *Boarding System* contract

Clause	Agent	Modality	Action	Reparation Clause	Time Restriction
0	*Passenger*	*Permission*	Go to the checkin desk (`checkin`)	∅	$t_0 - 120$
1	*Passenger*	*Obligation*	Present boarding pass (`PBP`)	∅	5
2	*Passenger*	*Obligation*	Show her passport (`ShP`)	∅	5
3	*Passenger*	*Permission*	Carry two hand luggage (`CToHL`)	∅	120
4	*Passenger*	*Permission*	Board (`board`)	6 & 15	90
5	*Passenger*	*Permission*	Board with hand luggage (`hl`)	6 & 15	10
6	*Airline Company*	*Obligation*	Allow passenger to board (`board`)	∅	90
7	*Passenger*	*Obligation*	Pass the security checkpoints (`PSC`)	∅	60
8	*Security Staff*	*Permission*	Carry detectors(`CD`)	∅	120
9	*Security Staff*	*Permission*	Examine passanger PC (`EPP`)	∅	120
10	*Passenger*	*Obligation*	Take out PC (`TOP`)	∅	10
11	*Passenger*	*Prohibition*	Taking risk articles (`TRA`)	∅	$t_{landing}$
12	*Passenger*	*Obligation*	Check in risk articles (`CRA`)	13	$t_0 - 120$
13	*Security Staff*	*Permission*	Requisition of risk articles(`RRA`)	∅	10
14	*Security Staff*	*Permission*	Deny access to boarding area(`DRA`)	∅	10
15	*Airline Company*	*Obligation*	Put her hand luggage in the hold (`hlhold`)	∅	20
16	*Airline Company*	*Obligation*	Allow passenger to board (`board`)	∅	90
17	*Passenger*	*Obligation*	Liquids of 100ml (`liquids`)	∅	120
18	*Passenger*	*Obligation*	Medication with receipt (`medication`)	∅	120
19	*Passenger*	*Obligation*	Demonstrate liquid medication (`liq`)	∅	120
20	*Passenger*	*Obligation*	Check in the firearms (`firearms`)	∅	$t_0 - 120$
21	*Passenger*	*Obligation*	Know her rights (`rights`)	∅	120
22	*Passenger*	*Permission*	File a complaint (`complaint`)	∅	120

$$
\begin{array}{lcl}
\text{timeout}(\top) & \stackrel{\text{df}}{=} & \infty \\
\text{timeout}(\bot) & \stackrel{\text{df}}{=} & \infty \\
\text{timeout}(\mathcal{P}_k(a)[d]) & \stackrel{\text{df}}{=} & d \\
\text{timeout}(\mathcal{O}_k(a)[d]) & \stackrel{\text{df}}{=} & d \\
\text{timeout}(\mathcal{F}_k(a)[d]) & \stackrel{\text{df}}{=} & d \\
\text{timeout}(\text{cond}_k(a)[d](\varphi_1, \varphi_2)) & \stackrel{\text{df}}{=} & d \\
\text{timeout}(\mathsf{wait}(d)) & \stackrel{\text{df}}{=} & d \\
\text{timeout}(\mathsf{rec}\ x.\varphi \mid x) & \stackrel{\text{df}}{=} & \text{timeout}(\varphi) \\
\text{timeout}(\varphi_1 \vee \varphi_2) & \stackrel{\text{df}}{=} & \min\{\text{timeout}(\varphi_1), \text{timeout}(\varphi_2)\} \\
\text{timeout}(\varphi_1 \wedge \varphi_2) & \stackrel{\text{df}}{=} & \min\{\text{timeout}(\varphi_1), \text{timeout}(\varphi_2)\} \\
\text{timeout}(\varphi_1; \varphi_2) & \stackrel{\text{df}}{=} & \text{timeout}(\varphi_1) \\
\text{timeout}(\varphi_1 \triangleright \varphi_2) & \stackrel{\text{df}}{=} & \text{timeout}(\varphi_1)
\end{array}
$$

And finally we obtain the result that we need: any timed transition taking less than the timeout of a contract preserves the structure of a contract.

Proposition 12. Given contract φ and time $t < \text{timeout}(\varphi)$, advancing φ by t time preserves the structure of the contract: if $\varphi \overset{t}{\rightsquigarrow} \varphi'$, then: $\varphi \equiv_s \varphi'$.

Proof. The proof is simple by structural induction. □

We can now define the closure of a contract φ as all formulae reachable from φ through action transitions and timeout time transitions.

Definition 20. We define the *closure* of a contract formula φ, written *closure*(φ), to be the set of all contract formulae reachable through a combination of visible action transitions and timeout transitions. Formally, *closure*(φ) is the smallest set such that: (i) $\varphi \in$ *closure*(φ); (ii) if $\varphi_1 \in$ *closure*(φ), and $\varphi_1 \xrightarrow{a,k} \varphi_2$, then $\varphi_2 \in$ *closure*(φ); and (iii) if $\varphi_1 \in$ *closure*(φ), and $\varphi_1 \overset{\text{timeout}(\varphi_1)}{\rightsquigarrow} \varphi_2$, then $\varphi_2 \in$ *closure*(φ).

It is easy to prove, that for a contract φ whose time constraints are non-zero constants, the closure of φ does not exhibit Zeno-like behaviour[3]. It also follows that the relations of timeout time steps and visible event steps are sufficient to characterize the operational semantics progress of a contract to a violation or otherwise.

8.2 A Monitoring Algorithm

The monitoring algorithm for our contract logic is shown in Algorithm 1. The state of the monitor is stored in variable *contract* while variable *systime* keeps track of the last timestamp processed by the system. Initially, these variables are set to ψ and 0 (line 1). The monitoring algorithm is effectively a loop (lines 2–17) which checks whether there was a violation upon every iteration. Upon entering the loop, any pending timer triggers are replaced (lines 3–5), enacting a process which creates a special *timeout* event (line 4) to be launched (asynchronously) after the current contract times out. In the meantime, execution is blocked until an event is received (line 6). If the event received is the *timeout* event, the monitored formula updated accordingly using the *timestep* function which returns the unique formula satisfying $\psi \overset{t}{\rightsquigarrow}$ *timestep*(φ, t) with the time advanced by *timeout*(ψ) time units (lines 7–10). If, however, the event received is a system event e with timestamp t,

3. By Zeno-like behaviour, we mean an infinite number of arbitrarily smaller time steps whose sum converges, thus blocking time from progressing.

the monitoring state is updated by first advancing time by $(t - systime)$ time units, and then stepping forward using the *step* function which returns the unique formula such that $\psi \xrightarrow{e} step(\psi, e)$ (lines 11–14). Finally, the *systime* variable is updated accordingly (line 16).

It is worth noting that the algorithm replicates the two types of transitions required to advance a contract: (i) maximally advancing time until the structure of the contract changes (the case of a timeout event); and (ii) processing a system event. This ensures that the state of the contract monitor advances steadily in correct steps (assuming that *timestep* and *step* correctly implement the rules from the semantics). Furthermore, progress is ensured since stepping along maximal time steps never results in Zeno-like behaviour.

```
1  contract = φ; systime = 0;
2  while ¬vio(contract) do
3  |  reset timer to timeout(contract))
4  |  |  createEvent(TIMEOUT);
5  |  end
6  |  switch getEvent() do
7  |  |  case TIMEOUT do
8  |  |  |  Δt = timeout(contract);
9  |  |  |  contract = timestep(contract, Δt);
10 |  |  end
11 |  |  case EVENT e WITH TIMESTAMP t do
12 |  |  |  Δt = t − systime;
13 |  |  |  contract = step(timestep(contract, Δt), e);
14 |  |  end
15 |  end
16 |  systime = systime + Δt;
17 end
18 report(VIOLATION);
```

Algorithm 1: Algorithm to monitor timed contracts

8.3 Runtime Verification Using Larva

Rather than programming directly Algorithm 1 from scratch, we will take advantage of an existing runtime verification tool. In particular, we will use Larva [49], which uses Dynamic Automata with Timers and Events (DATEs) as a specification language. DATEs are symbolic timed automata enriched in a number of aspects. The rest of this section introduces DATEs, with a number of adaptations over the original formalization to facilitate understanding and support the contract translation.

8.3.1 Preliminaries

There are three main elements in DATE transitions: (i) *Events* refer to observable actions which a DATE may react to. (ii) *Conditions* are boolean expressions taking into consideration both the DATE's symbolic state and the system state, which decide whether a transition is taken or not. (iii) *Actions* are modifications that are done to the DATE or system state upon observing an event and satisfying the condition. What follows formally defines these concepts.

Events which a DATE will be able to react to are either (i) system events over an alphabet SYSTEMEVENT, corresponding to control- or data-flow points of interest during the execution of the system; or (ii) timer events which are triggered upon a timer t reaching a threshold limit L written $t@L$. Timer limits can either take the form of a time constant $T \in \mathbb{T}$ (where $\mathbb{T}$ refers to the continuous time domain), or deadline variables $D \in \text{DEADLINE}$. Unlike constant limits, deadline variables can be dynamically modified during the traversal of the DATE.

$$\text{EVENT} ::= \text{SYSTEMEVENT} \mid \text{TIMER@}(\mathbb{T} \cup \text{DEADLINE})$$

DATE conditions and actions may also refer to the state of the system which is being monitored e.g. to react to a *login* event only if the system is in alert mode. The state of the system σ will be assumed to range over the type STATE_s. In addition, monitors may keep their own state e.g. the monitor may keep track of how many users are logged in, in order to react to a *login* only when more than 100 concurrent users are using the system. The symbolic DATE state μ will be assumed to range over the type STATE_m.

In addition, a DATE configuration will also keep track of the timer values $\tau \in \text{STATE}_T$, assigning a time value to each time such that $\text{STATE}_T = \text{TIMER} \to \mathbb{T}$. Similarly, it keeps track of the current value of the timer variable deadlines $\delta \in \text{STATE}_D$ where $\text{STATE}_D = \text{DEADLINE} \to \mathbb{T}$. We will abuse notation and write $\delta(L)$ to extend the function to work also on constant deadlines (in which case that constant deadline is returned) and $\tau + \Delta$ (where $\Delta \in \mathbb{T}$) to denote the timer state in which all timers are advanced by Δ time units. The symbolic state of a DATE is thus defined to be a combination of all these parts: $\text{STATE}^+_M = \text{STATE}_m \times \text{STATE}_T \times \text{STATE}_D$.

Conditions $c \in \text{CONDITION}$ are predicates over the system and full monitoring state:

$$\text{CONDITION} = (\text{STATE}_s \times \text{STATE}^+_M) \to \mathbb{B}$$

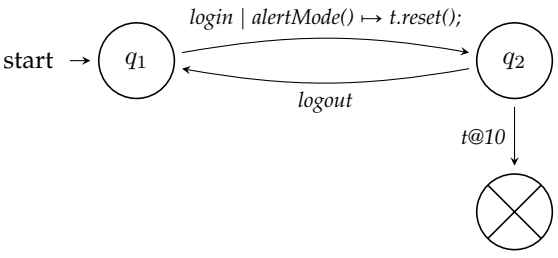
Similarly, actions $\alpha \in \text{ACTION}$ are functions which, based on the system state, may update any part of the full monitoring state:

$$\text{ACTION} = (\text{STATE}_s \times \text{STATE}^+_M) \to \text{STATE}^+_M$$

8.3.2 Formally Defining DATEs

A DATE is quadruple $\langle Q, q_0, K, B \rangle$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, $K \subseteq (Q \times \text{EVENT} \times \text{CONDITION} \times \text{ACTION} \times Q)$ is a set of event-condition-action transitions, and $B \subseteq Q$ is a set of bad states. We will write $q \xrightarrow{e|c \mapsto \alpha} q'$ to denote a transition $(q, e, c, \alpha, q') \in K$.

The following figure shows an example of DATE whereupon the detection of a login event on alert mode, starts a timer of ten minutes. If the ten minutes elapse before a logout, the DATE reaches a bad state.



The *event triggers* from a DATE state q, written $triggers(q)$, are defined to be all events which appear on transitions outgoing from q: $triggers(q) \stackrel{\text{df}}{=} \{e \mid \exists q', c, \alpha \cdot q \xrightarrow{e|c \mapsto \alpha} q'\}$.

The semantics of a DATE with dynamic timer deadlines can now be defined using this notation. The configuration of the monitor consists of (i) the state $q \in Q$ of the DATE; and (ii) the symbolic monitoring state $\sigma \in \text{STATE}^+_M$ of the monitor. Given a monitoring configuration, the *earliest timer trigger* is the least time which will trigger an outgoing timer event transition if no other event is received:

$$\mathit{earliest}(q,(\mu,\tau,\delta)) \stackrel{\text{df}}{=} \min\{\delta(L)-\tau(t) \mid t@L \in \mathit{triggers}(q) \wedge \delta(L) \geqslant \tau(t)\}$$

The semantics of DATEs will specify how the configuration of the DATE changes upon event triggering or time passing. We will have two forms of operational semantics relations: (i) $C \xrightarrow{e,\Delta,\sigma} C'$ to denote that the monitor moves from configuration C to C' upon the system receiving event e after Δ time units (from the last transition) and with the system state snapshot at that time being σ; (ii) $C \overset{\Delta,\sigma}{\rightsquigarrow} C'$ to denote that the monitor goes from configuration C to C' after Δ time units of inactivity at the end of which the system state is σ.

The first relation is defined with the implicit condition that an event e has triggered and another two preconditions: the existence of a transition triggering on e and the satisfaction of the condition c. The side-condition ensures that no timer-triggered transitions should have modified the configuration before.

$$\frac{q \xrightarrow{e|c\mapsto\alpha} q' \qquad c(\sigma,(\mu,\tau,\delta))}{(q,(\mu,\tau,\delta)) \xrightarrow{e,\Delta,\sigma} (q',\alpha(\mu,\tau+\Delta,\delta))}\ \Delta < \mathit{earliest}(q,(\mu,\tau,\delta))$$

The second relation is similar to the previous but, while not requiring the occurrence of a system event, requires that the timer has reached its deadline.

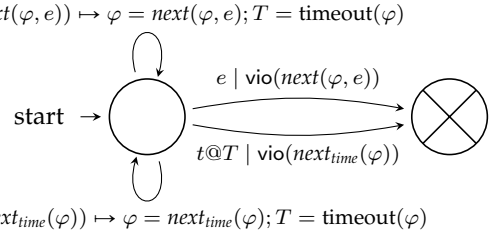
$$\frac{q \xrightarrow{t@L|c\mapsto\alpha} q' \qquad \tau(t)+\Delta=\delta(L) \qquad c(\sigma,(\mu,\tau,\delta))}{(q,(\mu,\tau,\delta)) \overset{\Delta,\sigma}{\rightsquigarrow} (q',\alpha(\mu,\tau+\Delta,\delta))}\ \Delta = \mathit{earliest}(q,(\mu,\tau,\delta))$$

Building on the earlier example, consider the case where the automaton is in the second state with the timer just reset: $(q_2,(\varnothing,t\mapsto 0,t\mapsto 10))$. If a logout event occurs after six minutes, then $\Delta = 6$, while $\mathit{earliest}(q_2,(\varnothing,t\mapsto 6,t\mapsto 10)) = 10-6 = 4$. Therefore the first relation would apply, updating the configuration to $(q_1,(\varnothing,t\mapsto 6,t\mapsto 10))$. On the other hand, if no logout event occurs within ten minutes, then $\mathit{earliest}(q_2,(\varnothing,t\mapsto 0,t\mapsto 10)) = 10-0 = 10$ causing the second relation to be applied resulting in the configuration $(q_\times,(\varnothing,t\mapsto 10,t\mapsto 10))$.

8.3.3 Implementation

The operational semantics given to the contracts, along with the timers and events, provide the framework to transform contracts in our calculus into DATEs which can be used to runtime verify the performance of parties involved.

We adapt the derivative-based [50] runtime verification algorithm shown in Algorithm 1 in order to obtain a DATE which reacts to the events appropriately. The resulting two state DATE keeps track of the current contract in a variable φ and updates triggers on either an event or the timeout to timeout(φ) triggers. Initially, φ is set to the contract to monitor while T to timeout(φ):



The function $\mathit{next}(\varphi,e)$ corresponds to $\mathit{step}(\mathit{timestep}(\varphi,\Delta t),e)$ while $\mathit{next}_{time}(\varphi)$ corresponds to $\mathit{timestep}(\varphi,\Delta t)$ — in both cases Δt is the time elapsed since the last processed event. Using this construction, a trace leads to the bad state of the DATE if and only if it violate the initial contract.

8.3.4 Practical Evaluation

To test our approach, we implemented the case study in Java with each action represented as a method call. When evaluated empirically, runtime verification tools, typically (e.g. [51]) get evaluated by comparing the time needed to run the system with and without monitoring. In this case, this is not practical since the system is simply a sequence of dummy method executions. Instead, the purpose of this quantitative evaluation is (i) to verify the correct behaviour of the monitor, i.e., that a violation is indeed reported when it actually occurs and vice versa; and (ii) to verify our intuition that the monitor will scale linearly with the size of the execution of the underlying system.

A number of test cases were generated, each representing the interactions of a single user involving a varying number of actions. In each case, the monitor verdict was as expected. Subsequently, different levels of traffic were generated by launching several users in parallel ranging from 100 user to 100,000 users. The experiment[4] was run on a laptop with an Intel i7-855U processor and 16GB RAM.

The results in Table 11 show that as the number of users increases, the CPU time per user stabilises at around 0.3 milliseconds. This confirms our reasoning that since the individual user monitors do not interact, the monitoring effort scales linearly to the number of users. One would only expect this trend to stop upon reaching a large number of users, the performance of an underlying framework starts deteriorating, e.g., the thread pool grows larger than what is efficiently manageable.

Regarding memory, since contract monitors only need to keep track of a state of bounded size per user per contract, this was not considered to be an issue.

9 Conclusions

In this paper we have extended the contract logic from [11] to deal with real time contracts. The resulting 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

4. The related code is available at: https://github.com/aarandag/larva-timedcontracts

Table 11
Experiment measurements rounded to 3 significant figures

Thousands of users	0.1	0.5	1	5	10	50	100
No monitoring (s)	0.0556	0.231	0.261	0.736	0.811	7.00	12.1
With monitoring (s)	0.104	0.433	0.595	2.16	4.05	18.1	38.4
Difference (s)	0.0480	0.202	0.334	1.43	3.24	11.1	26.4
Difference per user (ms)	0.480	0.404	0.334	0.285	0.324	0.223	0.264

a notion of similarity between contracts, which takes into account predicates over system states, and shown how these semantics can be used for runtime verification of contracts. Finally, we showed the utility of the timed calculus by applying it to an airline check-in desk case study. Our implementation of the monitoring engine worked correctly and within reasonable resource bounds.

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 [52], [53], [54] which could offer insight on how our work can be extended to build contract enforcement engines, a notion which has not been widely explored in the deontic logic world.

10 Acknowledgements

This research has been partially supported by the Spanish MINECO/FEDER project DArDOS (TIN2015-65845-C3-1-R and TIN2015-65845-C3-2-R) and the Comunidad de Madrid project SICOMORo-CM (S2013/ICE-3006).

References

[1] Georg Henrik Von Wright, Deontic Logic, Mind 60 (237) (1951) 1–15.
[2] S. Fenech, G. J. Pace, J. C. Okika, A. P. Ravn, G. Schneider, On the specification of full contracts, Electr. Notes Theor. Comput. Sci. 253 (1) (2009) 39–55.
URL http://dx.doi.org/10.1016/j.entcs.2009.09.027
[3] S. Khosla, System specification: A deontic approach, Ph.D. thesis, Imperial College of Science and Technology, University of London (1988).
[4] P. Jeremaes, S. Khosla, T. Maibaum, A modal (action) logic for requirements specification, Software Engineering 86 (1986) 278–294.
[5] N. Belnap, M. Perloff, In the realm of agents, Ann. Math. Artif. Intell. 9 (1-2) (1993) 25–48.
[6] J. F. Horty, Agency and Deontic Logic, Oxford University Press, 2001.
[7] G. Governatori, Z. Milosevic, Dealing with contract violations: formalism and domain specific language, in: EDOC Enterprise Computing Conference, Ninth IEEE International, IEEE Computer Society, 2005, pp. 46–57.
[8] G. J. Pace, F. Schapachnik, Contracts for Interacting Two-Party Systems, in: FLACOS'12, Vol. 94 of ENTCS, 2012, pp. 21–30. doi:10.4204/EPTCS.94.3.
[9] E. Asarin, V. Mysore, A. Pnueli, G. Schneider, Low dimensional hybrid systems - decidable, undecidable, don't know, Inf. Comput. 211 (2012) 138–159. doi:10.1016/j.ic.2011.11.006.
URL https://doi.org/10.1016/j.ic.2011.11.006
[10] Z. Chaochen, C. A. R. Hoare, A. P. Ravn, A calculus of durations, Inf. Process. Lett. 40 (5) (1991) 269–276. doi:10.1016/0020-0190(91)90122-X.
URL https://doi.org/10.1016/0020-0190(91)90122-X
[11] M. Cambronero, L. Llana, G. J. Pace, A calculus supporting contract reasoning and monitoring, IEEE Access 5 (2017) 6735–6745. doi:10.1109/ACCESS.2017.2696577.
URL https://doi.org/10.1109/ACCESS.2017.2696577
[12] J. Davies, S. Schneider, A brief history of timed CSP, Theor. Comput. Sci. 138 (2) (1995) 243–271. doi:10.1016/0304-3975(94)00169-J.
URL https://doi.org/10.1016/0304-3975(94)00169-J
[13] W. Yi, 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, 1991, pp. 217–228. doi:10.1007/3-540-54233-7_136.
URL https://doi.org/10.1007/3-540-54233-7_136
[14] C. Prisacariu, G. Schneider, A dynamic deontic logic for complex contracts, The Journal of Logic and Algebraic Programming 81 (4) (2012) 458 – 490, special Issue: NWPT 2009. doi:https://doi.org/10.1016/j.jlap.2012.03.003.
URL http://www.sciencedirect.com/science/article/pii/S1567832612000197
[15] A. Z. Wyner, Sequences, obligations, and the contrary-to-duty paradox, in: Deontic Logic and Artificial Normative Systems, 8th International Workshop on Deontic Logic in Computer Science, DEON 2006, Utrecht, The Netherlands, July 12-14, 2006, Proceedings, 2006, pp. 255–271. doi:10.1007/11786849_21.
URL https://doi.org/10.1007/11786849_21
[16] S. Azzopardi, G. J. Pace, F. Schapachnik, G. Schneider, Contract automata - an operational view of contracts between interactive parties, Artif. Intell. Law 24 (3) (2016) 203–243. doi:10.1007/s10506-016-9185-2.
URL https://doi.org/10.1007/s10506-016-9185-2
[17] G. Díaz, M. E. Cambronero, E. Martínez, G. Schneider, Specification and verification of normativetexts using C-O diagrams, IEEE Trans. Software Eng. 40 (8) (2014) 795–817. doi:10.1109/TSE.2013.54.
URL http://dx.doi.org/10.1109/TSE.2013.54
[18] J. M. Broersen, F. Dignum, V. Dignum, J. C. Meyer, Designing a deontic logic of deadlines, in: Deontic Logic in Computer Science, 7th International Workshop on Deontic Logic in Computer Science, DEON 2004, Madeira, Portugal, May 26-28, 2004. Proceedings, 2004, pp. 43–56. doi:10.1007/978-3-540-25927-5_5.
URL http://dx.doi.org/10.1007/978-3-540-25927-5_5
[19] J. B. Cole, J. Derrick, Z. Milosevic, K. Raymond, Author obliged to submit paper before 4 july: Policies in an enterprise specification, in: Proceedings of the International Workshop on Policies for Distributed Systems and Networks, POLICY, Springer-Verlag, London, UK, UK, 2001, pp. 1–17.
URL http://dl.acm.org/citation.cfm?id=646962.712118
[20] O. Marjanovic, Z. Milosevic, Towards formal modeling of e-contracts, in: Proceedings of the 5th IEEE International Conference on Enterprise Distributed Object Computing, EDOC, IEEE Computer Society, Washington, DC, USA, 2001, pp. 59–68.
[21] R. H. Marín, G. Sartor, Time and norms: a formalisation in the event-calculus, in: Proceedings of the Seventh International Conference on Artificial Intelligence and Law, ICAIL '99, Oslo,

Norway, June 14-17, 1999, 1999, pp. 90–99.
URL http://portal.acm.org/citation.cfm?id=323706.323719

[22] A. D. H. Farrell, M. J. Sergot, M. Sallé, C. Bartolini, Using the event calculus for tracking the normative state of contracts, Int. J. Cooperative Inf. Syst. 14 (2-3) (2005) 99–129. doi:10.1142/S0218843005001110.
URL https://doi.org/10.1142/S0218843005001110

[23] G. Governatori, A. Rotolo, A conceptually rich model of business process compliance, in: Conceptual Modelling 2010, Seventh Asia-Pacific Conference on Conceptual Modelling (APCCM 2010), Brisbane, Australia, January 18-21 2010., 2010, pp. 3–12.
URL http://crpit.com/confpapers/CRPITV110Governatori.pdf

[24] M. Hashmi, G. Governatori, M. T. Wynn, Modeling obligations with event-calculus, in: Rules on the Web. From Theory to Applications — 8th International Symposium, RuleML 2014, Co-located with the 21st European Conference on Artificial Intelligence, ECAI 2014, Prague, Czech Republic, August 18-20, 2014. Proceedings, 2014, pp. 296–310. doi:10.1007/978-3-319-09870-8_22.
URL https://doi.org/10.1007/978-3-319-09870-8_22

[25] N. Fornara, M. Colombetti, Specifying artificial institutions in the Event Calculus, IGI Global, 2009, Ch. 14, pp. 335–366.

[26] G. Governatori, A. Rotolo, G. Sartor, Temporalised normative positions in defeasible logic, in: The Tenth International Conference on Artificial Intelligence and Law, Proceedings of the Conference, June 6-11, 2005, Bologna, Italy, 2005, pp. 25–34. doi:10.1145/1165485.1165490.
URL http://doi.acm.org/10.1145/1165485.1165490

[27] M. Hashmi, G. Governatori, M. T. Wynn, Normative requirements for regulatory compliance: An abstract formal framework, Information Systems Frontiers 18 (3) (2016) 429–455. doi:10.1007/s10796-015-9558-1.
URL https://doi.org/10.1007/s10796-015-9558-1

[28] G. Governatori, A. Rotolo, Justice delayed is justice denied: Logics for a temporal account of reparations and legal compliance, in: Computational Logic in Multi-Agent Systems - 12th International Workshop, CLIMA XII, Barcelona, Spain, July 17-18, 2011. Proceedings, 2011, pp. 364–382. doi:10.1007/978-3-642-22359-4_25.
URL https://doi.org/10.1007/978-3-642-22359-4_25

[29] G. Governatori, J. Hulstijn, R. Riveret, A. Rotolo, Characterising deadlines in temporal modal defeasible logic, in: AI 2007: Advances in Artificial Intelligence, 20th Australian Joint Conference on Artificial Intelligence, Gold Coast, Australia, December 2-6, 2007, Proceedings, 2007, pp. 486–496. doi:10.1007/978-3-540-76928-6_50.
URL https://doi.org/10.1007/978-3-540-76928-6_50

[30] R. Gentilini, C. Piazza, A. Policriti, From bisimulation to simulation: Coarsest partition problems, J. Autom. Reasoning 31 (1) (2003) 73–103.

[31] R. J. van Glabbeek, B. Ploeger, Correcting a space-efficient simulation algorithm, in: CAV, 2008, pp. 517–529.

[32] C. Gregorio-Rodríguez, L. Llana, R. Martínez-Torres, Extending mcrl2 with ready simulation and iocos input-output conformance simulation, in: Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13-17, 2015, pp. 1781–1788. doi:10.1145/2695664.2695853.
URL http://doi.acm.org/10.1145/2695664.2695853

[33] A. Donzé, O. Maler, E. Bartocci, D. Nickovic, R. Grosu, S. A. Smolka, On temporal logic and signal processing, in: Automated Technology for Verification and Analysis - 10th International Symposium, ATVA 2012, Thiruvananthapuram, India, October 3-6, 2012. Proceedings, 2012, pp. 92–106. doi:10.1007/978-3-642-33386-6_9.
URL https://doi.org/10.1007/978-3-642-33386-6_9

[34] O. Maler, D. Nickovic, Monitoring properties of analog and mixed-signal circuits, STTT 15 (3) (2013) 247–268. doi:10.1007/s10009-012-0247-9.
URL https://doi.org/10.1007/s10009-012-0247-9

[35] S. Kochanthara, G. Nelissen, D. Pereira, R. Purandare, REVERT: runtime verification for real-time systems, in: 2016 IEEE Real-Time Systems Symposium, RTSS 2016, Porto, Portugal, November 29 - December 2, 2016, 2016, p. 365. doi:10.1109/RTSS.2016.044.
URL https://doi.org/10.1109/RTSS.2016.044

[36] U. Sammapun, I. Lee, O. Sokolsky, Rt-mac: Runtime monitoring and checking of quantitative and probabilistic properties, in: 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2005), 17-19 August 2005, Hong Kong, China, 2005, pp. 147–153. doi:10.1109/RTCSA.2005.84.
URL https://doi.org/10.1109/RTCSA.2005.84

[37] C. Colombo, G. J. Pace, G. Schneider, LARVA — safer monitoring of real-time java programs (tool paper), in: Seventh IEEE International Conference on Software Engineering and Formal Methods, SEFM, Hanoi, Vietnam, 23-27 November, 2009, pp. 33–37. doi:10.1109/SEFM.2009.13.
URL http://dx.doi.org/10.1109/SEFM.2009.13

[38] S. Azzopardi, G. J. Pace, F. Schapachnik, Contract automata with reparations, in: Legal Knowledge and Information Systems - JURIX: The Twenty-Seventh Annual Conference, Jagiellonian University, Krakow, Poland, 10-12 December, 2014, pp. 49–54. doi:10.3233/978-1-61499-468-8-49.
URL http://dx.doi.org/10.3233/978-1-61499-468-8-49

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

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

[41] M.-B. Airport, Air passenger rights, urlhttps://www.aeropuertomadrid-barajas.com/eng/air-passenger-rights.htm (2018).

[42] M.-B. Airport, Check-in counters, urlhttps://www.aeropuertomadrid-barajas.com/eng/checkin-madrid-airport.htm (2018).

[43] M.-B. Airport, Information about hand luggage, urlhttps://www.aeropuertomadrid-barajas.com/eng/regulations-hand-luggage.htm (2018).

[44] M. Leucker, C. Schallhart, A brief account of runtime verification, J. Log. Algebr. Program. 78 (5) (2009) 293–303. doi:10.1016/j.jlap.2008.08.004.
URL http://dx.doi.org/10.1016/j.jlap.2008.08.004

[45] C. Colombo, G. J. Pace, G. Schneider, Safe runtime verification of real-time properties, in: Formal Modeling and Analysis of Timed Systems, 7th International Conference, FORMATS 2009, Budapest, Hungary, September 14-16, 2009. Proceedings, 2009, pp. 103–117. doi:10.1007/978-3-642-04368-0_10.
URL http://dx.doi.org/10.1007/978-3-642-04368-0_10

[46] A. Bauer, M. Leucker, C. Schallhart, Monitoring of real-time properties, in: FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science, 26th International Conference, Kolkata, India, December 13-15, 2006, Proceedings, 2006, pp. 260–272. doi:10.1007/11944836_25.
URL http://dx.doi.org/10.1007/11944836_25

[47] B. Bonakdarpour, S. Navabpour, S. Fischmeister, Time-triggered runtime verification, Formal Methods in System Design 43 (1) (2013) 29–60. doi:10.1007/s10703-012-0182-0.
URL http://dx.doi.org/10.1007/s10703-012-0182-0

[48] J. A. Brzozowski, Derivatives of regular expressions, J. ACM 11 (4) (1964) 481–494. doi:10.1145/321239.321249.
URL http://doi.acm.org/10.1145/321239.321249

[49] C. Colombo, G. J. Pace, G. Schneider, Dynamic event-based runtime monitoring of real-time and contextual properties, in: Formal Methods for Industrial Critical Systems (FMICS), Vol. 5596 of Lecture Notes in Computer Science, L'Aquila, Italy, 2008, pp. 135–149.

[50] J. A. Brzozowski, Derivatives of regular expressions, J. ACM 11 (4) (1964) 481–494. doi:10.1145/321239.321249.
URL http://doi.acm.org/10.1145/321239.321249

[51] F. Chen, G. Rosu, Mop: an efficient and generic runtime verification framework, in: Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2007, October 21-25, 2007, Montreal, Quebec, Canada, 2007, pp. 569–588. doi:10.1145/1297027.1297069.
URL https://doi.org/10.1145/1297027.1297069

[52] Y. Falcone, T. Jéron, H. Marchand, S. Pinisetty, Runtime enforcement of regular timed properties by suppressing and delaying events, Sci. Comput. Program. 123 (2016) 2–41. doi:10.1016/j.scico.2016.02.008.
URL https://doi.org/10.1016/j.scico.2016.02.008

[53] F. B. Schneider, Enforceable security policies, ACM Trans. Inf. Syst. Secur. 3 (1) (2000) 30–50.

[54] S. Pinisetty, Y. Falcone, T. Jéron, H. Marchand, Tipex: A tool chain for timed property enforcement during execution, in: Runtime Verification - 6th International Conference, RV 2015 Vienna, Austria, September 22-25, 2015. Proceedings, 2015, pp. 306–320.

doi:10.1007/978-3-319-23820-3_22.
URL https://doi.org/10.1007/978-3-319-23820-3_22