TECHNICAL REPORT

Report No. CS2017-02
Date: September 2017

A Timed Contract-Calculus

María Emilia Cambronero
Luis Llana
Gordon J. Pace

Department of Computer Science
University of Malta
Msida MSD 06
MALTA

Tel: +356-2340 2519
Fax: +356-2132 0539
http://www.cs.um.edu.mt

A Timed Contract-Calculus

María Emilia Cambronero
University of Castilla-La Mancha, Spain.
MEmilia.Cambronero@uclm.es

Luis Llana
Universidad Complutense de Madrid, Spain.
llana@sip.ucm.es

Gordon J. Pace
University of Malta, Malta
gordon.pace@um.edu.mt

Abstract: *Over these past years, formal reasoning about contracts between parties participating in a transaction has been increasingly explored in the literature. There has been a shift of view from that viewing contracts simply as properties to be satisfied by the parties to contracts as first class syntactic objects which can be reasoned about independently of the parties' behaviour. In this paper, we present a contract calculus to reason about contracts abstracting the parties' behaviour in a simulation relation. We study the contracts evolution in time by associating time constraints with deontic clauses, which allows to associate time limit with permissions, obligations and prohibitions. Then, we show how the calculus can be used to support the runtime monitoring of contracts and apply it to a plane boarding system case study.*

A Timed Contract-Calculus*

María Emilia Cambronero
University of Castilla-La Mancha, Spain.
MEmilia.Cambronero@uclm.es

Luis Llana
Universidad Complutense de Madrid, Spain.
llana@sip.ucm.es

Gordon J. Pace
University of Malta, Malta
gordon.pace@um.edu.mt

Abstract: *Over these past years, formal reasoning about contracts between parties participating in a transaction has been increasingly explored in the literature. There has been a shift of view from that viewing contracts simply as properties to be satisfied by the parties to contracts as first class syntactic objects which can be reasoned about independently of the parties' behaviour. In this paper, we present a contract calculus to reason about contracts abstracting the parties' behaviour in a simulation relation. We study the contracts evolution in time by associating time constraints with deontic clauses, which allows to associate time limit with permissions, obligations and prohibitions. Then, we show how the calculus can be used to support the runtime monitoring of contracts and apply it to a plane boarding system case study.*

1 Introduction

The need for formal techniques for reasoning about contracts is becoming increasingly important as software systems interact more frequently with other software 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 such approaches do not address sufficiently well. Deontic logics have been developed precisely to deal with such a need to talk about *ideal* behavior of a system, possibly also including exceptional situations when the system deviates from such behavior. For instance, consider a contract which specifies that a party is to perform a particular action, but if they fail to do so, will incur an additional charge

*This work received financial support from the Spanish Government (cofinanced by FEDER funds) through the TIN2012-36812-C02-02 and TIN2015-65845-c03-02 Projects.

(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 [FPO+09]. 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 behavior. In interacting systems, contracts play an even more important role, since an agent's behavior (or non-behavior) directly impacts other agents. Surprisingly, most contract logics reason about deontic modalities such as obligations and permissions per agent, and there is limited work on reasoning about directed deontic modalities [GM05, PS12], 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 or contract. Over these past couple of decades, however, there has been a shift of view, distinguishing properties from contracts — moving from a view of e-contracts simply as properties to be satisfied by the agents involved in the contract, to contracts as first class syntactic objects which can be reasoned about independently of the agents' behavior. In much of the literature, however, contract comparison is still defined by quantifying over all possible behavior of the systems, making reasoning about contracts still depends directly on contract satisfaction and violation predicates parametrised by the behavior they are regulating.

In a previous version [CLP17] a calculus to reason about contracts independent of the systems was presented, however time was not considering. In this paper, we present a timed calculus to reason about contracts abstracting away the agents' behavior in the simulation relation and considering deontic modalities and conditions with time constraints. Then, we give an operational view of contracts, using notions from process calculi to model the notion of contracts, and enabling their analysis and comparison using bisimulation techniques on their operational behavior. The approach also enables us to reason about nondeterminism in contracts. We complete this formalism by adding time in order to predict the contract accomplishments according to time.

The paper is organised as follows. In section 3, we present the notation we will use to formalize our notions. We then present our timed contract calculus in section 4 and formalise the notion of refinement of contracts in section 5. Furthermore, we show how we

can transform contracts written in our calculus into runtime monitors which report violation in section 6. We show how the calculus can be applied to a standard case study of a Plane Boarding System contract between a passenger and the airline company in section 7. We compare our approach to related work in section 2 and conclude in section 8.

2 Related Work

There is a long history of contract formalisation in terms of deontic logics. The timed calculus we present in this paper is based on an untimed contract calculus we presented in [CLP17]. 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 standard notions 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 [CLP17]. In this paper we will focus solely on related work from the literature formalising time in contracts. The way we augment the calculus with time shares much how real-time is typically augmented in process calculi e.g. timed CSP [DS95] and timed CCS [Yi91] — 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 surprisingly little work formalising the notions behind explicitly timed contracts. Even when allowing for temporal modalities, many deontic logics and contract languages (e.g. [PS09, Wyn06, APSS16]) 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*). In literature some proposals considering explicit time in contracts can be found. Among them some remarkable works are [DCMS14, BDDM04, CDMR01].

In citeTSE14 a graphical representation of contracts is presented. In this work the authors define diagrams to represent the 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 operational semantics of this formalism is not presented and they consider the parties independently.

J. Broersen et al. [BDDM04] 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 deal with time using only 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 deadline obligations 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.

In [CDMR01] J. Cole et al. present a case study of a conference programme to illustrate how policies (also covering obligations, prohibitions and permissions) might be formulated and refined alongside the refinement of the system specification. They model timed action and events, but they do not give a formal semantics for this purpose.

One of the richest real-time deontic logics was developed by Marjanovic et al. [MM01]. The logic presented covers various ways in which temporal constraints can be added within contracts. Although they present some ideas of how such constraints can be verified for the conjunction of different clauses, the logic lacks notions such as contract sequentiality and reparation which are able to handle in our logic.

3 Notation

Contracts regulate the behavior 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 behavior in order to be able to formalise the notion of contracts in the following sections.

We will assume that the underlying system consists of a number of indexed agents running in parallel: $||_{i=1}^{n} A_i$, where each A_i is an agent indexed by $i \in \mathcal{I}$ with $\mathcal{I}$ being the index set. We will use variables A, A' and indexed versions for individual agent behavior. We also use $\mathcal{A}$, $\mathcal{A}'$, etc. to denote the system as a whole. The system will be assumed to perform actions over alphabet Act. We will write $A \xrightarrow{a} A'$ to indicate that agent A can perform action $a \in \mathsf{Act}$ to become A' and $A \xrightarrow{a}\!\!\!\!\!/\;\;$ to mean that A cannot perform action a: $\neg\exists A' \cdot A \xrightarrow{a} A'$.

We can now define $\mathcal{A} \xrightarrow{a,s} \mathcal{A}'$ to denote that system $\mathcal{A}$ can perform action $a \in \mathsf{Act}$ to become system $\mathcal{A}'$ with s denoting the set of parties involved in the transition. This is formally defined as follows:

Definition 1 *Transitions Notation.*

- $\mathcal{A} \xrightarrow{a,\mathsf{only(l)}} \mathcal{A}'$ *means that agent* l *can perform action* a *but no other agent can synchronize with it. Formally, we define* $\mathcal{A} \xrightarrow{a,\mathsf{only(l)}} \mathcal{A}'$ *as: (i) only agent* l *has evolved:* $\mathcal{A} = A_1 \parallel \cdots \parallel A_l \parallel \cdots \parallel A_n$ *and* $\mathcal{A}' = A_1 \parallel \cdots \parallel A'_l \parallel \cdots \parallel A_n$*; (ii)* A_l *has evolved with action* a*:* $A_l \xrightarrow{a} A'_l$*; and (iii) no other agent could have synchronised on action* a*:* $\forall k \neq l \cdot A_k \xrightarrow{a}\!\!\!\!\!/\;$.
- $\mathcal{A} \xrightarrow{a,\{l,k\}} \mathcal{A}'$ *means that agents* l *and* k *synchronize on action* a*. Formally, we define* $\mathcal{A} \xrightarrow{a,\{l,k\}} \mathcal{A}'$ *to mean that: (i) only agents* k *and* l *have evolved over action* a*:* $\mathcal{A} = A_1 \parallel \cdots \parallel A_l \parallel \cdots \parallel A_k \parallel \cdots \parallel A_n$*, and* $\mathcal{A}' = A_1 \parallel \cdots \parallel A'_l \parallel \cdots \parallel A'_k \parallel \cdots \parallel A_n$*; and (ii) both agents* l *and* k *have evolved over actions* a*:* $A_l \xrightarrow{a} A'_l$ *and* $A_k \xrightarrow{a} A'_k$*.*
- $\mathcal{A} \rightsquigarrow^{t} \mathcal{A}'$ *means that the system* $\mathcal{A}$ *evolves to* $\mathcal{A}'$ *when* t *time units pass.*

We also write $\mathcal{A} \xrightarrow{a,l}$ to denote that agent l is capable of performing action a: $\exists A' : A_k \xrightarrow{a} A'$, and $\mathcal{A} \xrightarrow{a,l}\!\!\!\!\!/\;$ for its negation.

In order to formalise the notion of violation of contracts, we will use predicates over agent behaviour.

Definition 2 *Let* k *be an agent index and* $a \in \mathsf{Act}$*. A predicate is defined as:*

$$P ::= tt \mid ff \mid (a,k) \mid \overline{(a,k)} \mid \neg(a,k) \mid P \vee Q \mid P \wedge Q$$

The set of predicates is denoted by $\mathcal{P}$*.*

Predicates tt and ff denote true and false respectively. The predicate $\overline{(a,k)}$ indicates that agent k wants to perform the action a, but this action is not offered by any other agent for synchronization. Predicate disjunction and conjunction is indicated by $P \vee Q$ and $P \wedge Q$.

Definition 3 *Let* $\mathcal{A}$ *and* $\mathcal{A}'$ *be two systems and* P_1 *and* $P_2 \in \mathcal{P}$*. Formally, the semantics of predicates are defined as:*

$$\begin{array}{lcl}
\mathcal{A} \vdash tt & \stackrel{df}{=} & true \\
\mathcal{A} \vdash ff & \stackrel{df}{=} & false \\
\mathcal{A} \vdash (a,k) & \stackrel{df}{=} & \exists s,\ \mathcal{A}' \cdot \mathcal{A} \xrightarrow{a,s} \mathcal{A}' \wedge k \in s \\
\mathcal{A} \vdash \overline{(a,k)} & \stackrel{df}{=} & \exists \mathcal{A}' \cdot \mathcal{A} \xrightarrow{a,\mathsf{only(k)}} \mathcal{A}' \\
\mathcal{A} \vdash \neg(a,k) & \stackrel{df}{=} & \nexists s,\ \mathcal{A}' \cdot \mathcal{A} \xrightarrow{a,s} \mathcal{A}' \wedge k \in s \\
\mathcal{A} \vdash P_1 \vee P_2 & \stackrel{df}{=} & \mathcal{A} \vdash P_1 \ or\ \mathcal{A} \vdash P_2 \\
\mathcal{A} \vdash P_1 \wedge P_2 & \stackrel{df}{=} & \mathcal{A} \vdash P_1 \ and\ \mathcal{A} \vdash P_2
\end{array}$$

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 $P \vdash Q$, iff for any system $\mathcal{A}$ for which $\mathcal{A} \vdash P$ holds, $\mathcal{A} \vdash Q$ also holds. We say that P is* equivalent *to Q, written $P \sim Q$, iff $P \vdash Q$ and $Q \vdash P$.*

4 A Timed Contract Calculus

We can now define our contract calculus to allow for an operational view of contracts. We start by defining the contract calculus syntax and an equivalence relation over the syntactic forms. We then define the notion of contract violation conditions based on which we give an operational semantics to the calculus. In this section, we will assume a time domain ranging over the non-negative reals: $\mathbb{T} = \mathbb{R}^+$.

4.1 Contract Syntax

Definition 5 *Well formed formulae φ in our contract calculus follows this syntax:*

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

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

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 obligations, permissions 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 action $\mathsf{wait}(d)$ represents a delay of d time units. 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: $\mathsf{wait}(2); \mathcal{O}_k(a)[3]$.
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 one must first satisfy the first contract φ_1 and then the second φ_2.

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 $[a, k, d](\varphi_1, \varphi_2)$ is a conditional contract which behaves like φ_1 if (a, k, d) holds on the system (i.e., if party k can perform action a within d time units), and like φ_2 otherwise. Note that we can generalise 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 perform action a in d time units, repeatedly. For simplicity, we will assume that (i) variables x will only occur under a recursive definition on that variable; and (ii) recursive variables are preceded by a prefix. In the rest of the paper, the set of contracts will be denoted by $\mathcal{C}$.

4.2 Syntactical Equivalence

In order to simplify the presentation of the operational semantics, we will define a syntactical equivalence relation $\equiv$ between well-formed formulae in the contract calculus. This equivalence relation must be applied on a well-formed formula and its subformulae before the rules of the operational semantics. It is defined as the least equivalence 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. $[a, k, 0](\varphi, \psi) = \psi$	

The $\equiv$ relation can be seen as rewriting rules read from left to right. We will write $\varphi \hookrightarrow \varphi'$ if φ' is the result of applying one of the equivalence rules from left to right on a subexpression of φ. We can show that $\hookrightarrow$ is terminating and confluent.

Proposition 1 *The syntactic equivalence relation applied from left to right is (i) terminating: there are no infinite sequences φ_1, $\varphi_2 \ldots$, such that for all values of i, $\varphi_i \hookrightarrow \varphi_{i+1}$; (ii) local confluent: if $\varphi \hookrightarrow \varphi_1$ and $\varphi \hookrightarrow \varphi_2$ then there is a contract φ' such that $\varphi_1 \hookrightarrow^* \varphi'$ and $\varphi_2 \hookrightarrow^* \varphi'$. 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. To prove local confluence, we perform case analysis on the different rules, which are applied to the subformulae to show that the confluence result holds.*

Based on these results, we can prove confluence using Newman's Lemma [New42].

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 $\varphi_2 \hookrightarrow^* \varphi'$.*

Given confluence and termination, we will write $\varphi \longmapsto \psi$ to mean than (i) φ can be syntactically reduced to ψ in a number of steps: $\varphi \hookrightarrow^* \psi$; and (ii) ψ cannot be reduced any further: $\psi \not\hookrightarrow$.

Definition 6 *Let us consider the relation $\mathcal{R} \subseteq \mathcal{C} \times \mathcal{C}$ defined as follows*

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

The structural equivalence congruence*, denoted by $\equiv_s$ is the smaller congruence that contains $\mathcal{R}$.*

4.3 Contract Violation

We can now formally define the notion of contract violation.

Definition 7 *We say that a contract φ is in a* violated *state, written $\mathsf{vio}(\varphi)$ if the contract has already been violated:*

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

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 party holding the permission wants to perform the action but is not offered a synchronizing action. In case of an obligation and prohibition, violation can only occur after the next action, and is thus not immediately violated. Immediate violations of conjunctions and disjunctions follow as expected.

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

Table 1: Obligation transition rules.

In the case of a reparation $\mathsf{vio}(\varphi \blacktriangleright \varphi')$, a violation can only occur, if φ, but also its reparation, are violated. In the case of $\mathsf{vio}([a,k,d](\varphi,\varphi'))$, 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 correct since the recursion is always guarded.

4.4 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; 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) $\varphi \rightsquigarrow^{d} \varphi'$ to represent that contract φ can evolve to contract φ' when d time units pass. In the following we will write $\varphi \xrightarrow{\alpha} \varphi'$ when α can be a tuple (a,k) or a tuple $\overline{(a,k)}$.

The core of any contract reasoning formalism are the rules defining the deontic modalities. The semantics of these modalities in our calculus are defined as follows:

Rules **O1**, **O2**, **O3** and **O4** (Table 1) define the behavior of $\mathcal{O}_k(a)[d]$, i.e., the obligation on agent k to perform action a within d time units. Rule **O1** handles the case of the obligation clause being satisfied when agent k does a within d time units, in this case, the contract reduces to the trivially satisfied one $\top$. Rule **O2** considers the case when another actor l performs an action b, leaving the obligation intact. Rule **O3** refers the case when an action b is not offered by any other part than l, then the obligation remains the same. Finally, **O4**

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

Table 2: Prohibition 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)	$\mathcal{P}_k(a)[d] \rightsquigarrow^{d'} \mathcal{P}_k(a)[d-d'], \quad d' \leq d$

Table 3: Permission transition rules.

(wait1)	$\mathsf{wait}(d) \rightsquigarrow^{d'} \mathsf{wait}(d-d'), \quad d' \leq d$
(wait2)	$\mathsf{wait}(d) \xrightarrow{\alpha} \mathsf{wait}(d)$

Table 4: Wait transition rule.

handles the case when d' time units pass with $d' \leq d$, then the obligation remains, but the obligation time decreases in d' time units.

Rules $\mathbf{F1}$, $\mathbf{F2}$, $\mathbf{F3}$ and $\mathbf{F4}$ (Table 2) define the cases for prohibition similar to obligation.

Permission of agent k to perform action a within d time units ($\mathcal{P}_k(a)[d]$) is defined through Rules $\mathbf{P1}$, $\mathbf{P2}$, $\mathbf{P3}$ and $\mathbf{P4}$ (Table 3). Rule $\mathbf{P1}$ 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 $\top$. $\mathbf{P2}$ handles the case when agent other than k perform an action, leaving k's permission intact, while $\mathbf{P3}$ handles the case when the permission is

(C1)	$[a,k,d](\varphi,\psi) \xrightarrow{a,k} \varphi$
(C2)	$[a,k,d](\varphi,\psi) \xrightarrow{b,l} \psi, \quad b,l \neq a,k$
(C3)	$[a,k,d](\varphi,\psi) \overset{d'}{\rightsquigarrow} [a,k,d-d'](\varphi,\psi), \quad d' \leq d$
(C4)	$[a,k,d](\varphi,\psi) \xrightarrow{\overline{(b,l)}} [a,k,d](\varphi,\psi), d > 0$

Table 5: Condition transition rules (I).

violated because agent k intended to perform action a, but it was not offered a synchronizing action. Finally, $\mathbf{P4}$ considers the case when d' time units elapse, with $d' \leq d$, then the permission remains, but the permission time decreases in d' time units.

The wait rules are presented in Table 4, which define two possible cases: when d' time units pass with $d' \leq d$, then the time delay of the $wait$ action decreases in d' time units (rule $\mathbf{wait1}$), and when the time delay remains because another thing happens (rule $\mathbf{wait2}$).

The rules for conditional contracts (Table 5) handle the cases when the condition holds ($\mathbf{C1}$), and when it does not ($\mathbf{C2}$), resolving the contract to the appropriate branch. The rule $\mathbf{C3}$ considers the case when the time passes in d' time units, with $d' \leq d$, in this case the conditional time decreases in d' time units. And finally, the rule $\mathbf{C4}$ handle the case when an action a is not offered by agent k, then the contract is not affected, and remains the same.

The rules for conjunction and disjunction (Table 6) are structurally identical, since both take the two contracts to evolve concurrently. The difference between the two operators is only exhibited in the cases of one of the two operands reduces to $\top$ or $\bot$, in these cases, the different equivalence rules reduce the formula in different ways (see subsection 4.2). The first rule $\mathbf{AO1}$ handles the case in which any actions are observed in both contracts, φ and ψ, then they evolve to φ' and ψ', respectively, therefore the conjunction or disjunction of them evolve in the same. The equivalence rules distinguish between the two operators, by allowing different reductions when one of the two operands is violated or is satisfied. The second rule $\mathbf{AO2}$ considers the case in which d time units pass for both contracts. Rules $\mathbf{AO3}$ shows the case in which d time units pass for the first contract, φ, then it evolves to $\top$ and d' for the second one, ψ, then it evolves to ψ', with $d' \geq d$, thus the contracts conjunction evolves as the second one; $\mathbf{AO4}$ 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 $\bot$, with $d \geq d'$, thus the contracts conjunction evolves as the first one. Rules $\mathbf{AO5}$ and $\mathbf{AO6}$ consider the cases in which the first or second contract have been already

(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)	$\dfrac{\varphi \overset{d}{\rightsquigarrow} \top,\ \psi \overset{d'}{\rightsquigarrow} \psi',\ d' \geq d}{\varphi \wedge \psi \overset{d'}{\rightsquigarrow} \psi'}$
(AO4)	$\dfrac{\varphi \overset{d}{\rightsquigarrow} \varphi',\ \psi \overset{d'}{\rightsquigarrow} \top,\ d \geq d'}{\varphi \wedge \psi \overset{d}{\rightsquigarrow} \varphi'}$
(AO5)	$\dfrac{\varphi \overset{d}{\rightsquigarrow} \bot,\ \psi \overset{d'}{\rightsquigarrow} \psi',\ d' \geq d}{\varphi \vee \psi \overset{d'}{\rightsquigarrow} \psi'}$
(AO6)	$\dfrac{\varphi \overset{d}{\rightsquigarrow} \varphi',\ \psi \overset{d'}{\rightsquigarrow} \bot,\ d \geq d'}{\varphi \vee \psi \overset{d}{\rightsquigarrow} \varphi'}$

Table 6: Transition rules for conjunction and disjunction ($\mathsf{op} \in \{\vee, \wedge\}$)

violated and how the disjunction of both contracts evolve, in an analogous manner as the conjunction.

The rules for reparation and sequential composition are presented in Table 7. The first two rules $\mathbf{V1}$ and $\mathbf{V2}$ 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 $\mathbf{S1}$ and $\mathbf{S2}$ 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 $\mathbf{V3}$ and $\mathbf{S3}$ are necessary for time additivity with reparation and sequential composition, respectively.

The final rules (Table 8) deal with recursion in a standard manner. Note that we are only considering guarded variables.

We can prove the following proposition indicating that any contract reacts to any action of

(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)	$\dfrac{\varphi \overset{d}{\rightsquigarrow} \top, \ \psi \overset{d'}{\rightsquigarrow} \psi'}{\varphi ; \psi \overset{d+d'}{\rightsquigarrow} \psi'}$

Table 7: Recover and sequential transitions rules

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

Table 8: Recursion transitions

a system, except for the $\top$ and $\bot$ contracts.

Proposition 2 *Given a contract $\varphi \in \mathcal{C}$, one of the following has to hold: (i) $\varphi \equiv \top$; (ii) $\varphi \equiv \bot$; or (iii) $\forall a \in \mathsf{Act},\ k \in \mathcal{I} \cdot \varphi \xrightarrow{a,k}$.*

Using these notions of basic contracts, we can define the notion of obligations and prohibitions which persist until a condition holds. The obligation of agent k to perform an action a within d timed units until a condition ($[b, l]$) holds (written $\mathcal{O}_k(a)[d]\mathcal{U}[b, l]$) and the prohibition of agent k from performing an action a within d timed units until a condition ($[b, l]$) holds (written $\mathcal{F}_k(a)[d]\mathcal{U}[b, l]$), where the condition $[b, l]$ means that the action b made by agent l is observed. Therefore, they are defined below:

$$\mathcal{O}_k(a)[d]\mathcal{U}[b, l] \stackrel{\text{df}}{=} \mathsf{rec}\ x.\mathcal{O}_k(a)[d] \wedge [b, l](\top, x)$$
$$\mathcal{F}_k(a)[d]\mathcal{U}[b, l] \stackrel{\text{df}}{=} \mathsf{rec}\ x.\mathcal{F}_k(a)[d] \wedge [b, l](\top, x)$$

In the rest of the paper, we will also use the notion of the longest time which may elapse before a contract changes structurally, which we refer to as the *timeout* of a contract.

Definition 8 *The* timeout *of a contract φ, written timeout(φ), gives the time which must elapse without any action occurring before the contract changes structurally, and timeout$(\varphi) \stackrel{df}{=}$ timeout$'(\varphi')$ where φ' is the unique final contract such that $\varphi \hookrightarrow^* \varphi'$, and timeout$'$ is inductively defined as follows:*

timeout$'(\top)$	$\stackrel{df}{=}$	∞	timeout$'(\bot)$	$\stackrel{df}{=}$	∞
timeout$'(\mathcal{P}_k(a)[d])$	$\stackrel{df}{=}$	d	timeout$'(\mathcal{O}_k(a)[d])$	$\stackrel{df}{=}$	d
timeout$'(\mathcal{F}_k(a)[d])$	$\stackrel{df}{=}$	d	timeout$'([a, k, d](\varphi_1, \varphi_2))$	$\stackrel{df}{=}$	d
timeout$'(\mathsf{wait}(d))$	$\stackrel{df}{=}$	d	timeout$'(\mathsf{rec}\ x.\varphi \mid x)$	$\stackrel{df}{=}$	timeout(φ)
timeout$'(\varphi_1 \vee \varphi_2)$	$\stackrel{df}{=}$	$\min\{$timeout(φ_1), timeout$(\varphi_2)\}$	timeout$'(\varphi_1; \varphi_2)$	$\stackrel{df}{=}$	timeout(φ_1)
timeout$'(\varphi_1 \wedge \varphi_2)$	$\stackrel{df}{=}$	$\min\{$timeout(φ_1), timeout$(\varphi_2)\}$	timeout$'(\varphi_1 \blacktriangleright \varphi_2)$	$\stackrel{df}{=}$	timeout(φ_1)

Any timed transition taking less than the timeout of a contract preserves the structure of a contract.

Proposition 3 *Given contract φ and time $t <$ timeout(φ), advancing φ by t time preserves the structure of the contract: if $\varphi \overset{t}{\rightsquigarrow} \varphi'$, then $\varphi \equiv_s \varphi'$.*

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

Definition 9 *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 is finite. In the rest of the paper, we will use the following notation.

Definition 10 *We write $\psi \xrightarrow{P}_{full} \psi'$ if there exist ψ_0 to ψ_{n+1} and P_0 to P_n such that (i) $\psi = \psi_0 \xrightarrow{P_0} \psi_1 \xrightarrow{P_1} \ldots \xrightarrow{P_n} \psi_{n+1} = \psi'$; (ii) there exists no P_{n+1} and ψ'' such that $\psi_{n+1} \xrightarrow{P_{n+1}} \psi''$; and (iii) $P \equiv P_0 \wedge P_1 \wedge \ldots P_n$. Note that, if there is no outgoing predicate transition from ψ, we get $\psi \xrightarrow{tt}_{full} \psi$.*

4.5 Contracts and Systems

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

Definition 11 *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 — through the following three rules:*

$$\mathbf{M1}\ \frac{\varphi \xrightarrow{a,k} \varphi',\ \mathcal{A} \xrightarrow{a,s} \mathcal{A}'}{\varphi \parallel \mathcal{A} \Rightarrow \varphi' \parallel \mathcal{A}'}\ k \in s \qquad \mathbf{M2}\ \frac{\varphi \xrightarrow{\mathsf{vio}(\varphi)} \bot,\ \mathcal{A} \vdash \mathsf{vio}(\varphi)}{\varphi \parallel \mathcal{A} \Rightarrow \bot \parallel \mathcal{A}}$$

$$\mathbf{M3}\ \frac{\mathcal{A} \xrightarrow{a,s} \mathcal{A}'}{\varphi \parallel \mathcal{A} \Rightarrow \varphi \parallel \mathcal{A}'}\ a \notin \mathsf{Act}$$

$$\mathbf{M4}\ \frac{\mathcal{A} \overset{d}{\rightsquigarrow} \mathcal{A}',\ \varphi \overset{d}{\rightsquigarrow} \varphi',\ \forall d' < d : \mathcal{A} \overset{d'}{\rightsquigarrow} \mathcal{A}'',\ \varphi \overset{d'}{\rightsquigarrow} \varphi'' : \mathcal{A}'' \not\vdash \mathsf{vio}(\varphi'')}{\varphi \parallel \mathcal{A} \overset{d}{\rightsquigarrow} \varphi' \parallel \mathcal{A}'}$$

$$\mathbf{M5}\ \frac{\mathcal{A} \parallel \varphi \overset{d}{\rightsquigarrow} \mathcal{A}' \parallel \varphi',\ \mathcal{A}' \vdash \mathsf{vio}(\varphi'),\ \mathcal{A}' \parallel \psi \overset{d'}{\rightsquigarrow} \mathcal{A}'' \parallel \psi'}{\mathcal{A} \parallel \varphi \blacktriangleright \psi \overset{d+d'}{\rightsquigarrow} \mathcal{A}'' \parallel \psi'}$$

Rule $\mathbf{M1}$ 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. Rule $\mathbf{M2}$ considers contract internal actions which, to be taken, require the condition predicate to be satisfied by the system. Rule $\mathbf{M3}$ handles actions on the system which the contract is not interested in.

Rule $\mathbf{M4}$ shows that time can pass unless there is a violation before. Finally, rule $\mathbf{M5}$ considers the case of the contract and system evolution when the system violates the contract, but its reparation is defined.

Based on these rules, we can define what it means for a system to break a contract.

Definition 12 *A system $\mathcal{A}$ is said to* currently break *contract $\varphi \in \mathcal{C}$, written $break_0(\mathcal{A}, \varphi)$, if $\mathcal{A} \vdash \mathsf{vio}(\varphi)$.*

A system $\mathcal{A}$ is said to break *contract $\varphi \in \mathcal{C}$, written $break(\mathcal{A}, \varphi)$ if there exists a computation that breaks the contract: $\exists \varphi', \mathcal{A}' \cdot \varphi \parallel \mathcal{A} \Rightarrow \varphi' \parallel \mathcal{A}' \wedge break_0(\mathcal{A}', \varphi')$.*

5 Refinement

Definition 13 *Let $\varphi, \psi \in \mathcal{C}$ and $R \subseteq \mathcal{C} \times \mathcal{C}$, we say that R is a* $\perp$-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 hold:

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

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

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

Definition 14 *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 $\varphi \overset{d}{\rightsquigarrow} \varphi'$ then one of the following conditions hold:

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

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

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

Definition 15

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

2. *Two contracts $\psi, \varphi \in \mathcal{C}$ are* $\top$-equivalent (respectively $\perp$-equivalent)*, written $\varphi \backsim_\top \psi$ (respectively $\varphi \backsim_\perp \psi$), iff $\varphi \preccurlyeq_\top \psi$ and $\psi \preccurlyeq_\top \varphi$ (respectively $\varphi \preccurlyeq_\perp \psi$ and $\psi \preccurlyeq_\perp \varphi$).*

Example 1

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

We now prove that contract simulation corresponds to contract strictness.

Theorem 1 *Let $\mathcal{A}$ be 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: break$(\mathcal{A}, \varphi) \Rightarrow$ break$(\mathcal{A}, \psi)$. Proof. Since $\varphi \preccurlyeq_\perp \psi$ then there is a simulation contract R such that $(\varphi, \psi) \in R$. On the other hand, since break$(\mathcal{A}, \varphi)$ there is a sequence of transitions $\varphi \parallel \mathcal{A} \stackrel{w}{\Longrightarrow} \psi' \parallel \mathcal{A}'$ where break$_0(\mathcal{A}', \varphi')$. This sequence of transitions can be unfolded into a computation of the following form:*

$$\varphi \parallel \mathcal{A} = \varphi_0 \parallel \mathcal{A}_0 \xrightarrow{\alpha_1} \varphi_1 \parallel \mathcal{A}_1 \xrightarrow{\alpha_2} \ldots \varphi_n \parallel \mathcal{A}_n \xrightarrow{\alpha_n} \varphi_n \parallel \mathcal{A}_n = \varphi' \parallel \mathcal{A}'$$

where $n \geq 0$. We are going to prove that we can simulate that computation beginning with the contract ψ_0 such that $(\varphi_k, \psi_k) \in R$ for any $0 \leq k \leq n$ by induction on n. If $n = 0$ the proof is immediate, so let us consider the inductive case $n > 0$. Let us consider the first transition. There are two cases:

Case 1: *$\varphi_0 \xrightarrow{a,k} \varphi_1$, $\mathcal{A}_0 \xrightarrow{a,s} \mathcal{A}_1$ with $k \in s$. Since $(\varphi_0, \psi_0) \in R$ then there is a contract ψ_1 such that $\psi \xrightarrow{a,k} \psi_1$ and $(\varphi_1, \psi_1) \in R$. So 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.*

Case 2: *$\varphi_0 \xrightarrow{P_1} \varphi$‘ where $\mathcal{A} \vdash P$. In this case $\mathcal{A}_0 = \mathcal{A}_1$. Since $(\varphi_0, \psi_0) \in R$ we deduce that either $(\varphi_1, \psi_0) \in R$ or there is a predicate P_1 and a contract ψ_1 such that $\psi_0 \xrightarrow{P_1'} \psi_1$ and $P_1 \vdash P_1'$. $\mathsf{vio}(\varphi) \vdash \mathsf{vio}(\psi)$, so $\mathcal{A} \vdash \mathsf{vio}(\psi)$. Therefore the transition $\psi \parallel \mathcal{A} \Rightarrow \psi' \parallel \mathcal{A}$ is possible. And we obtain the result by induction.*

Case 3: *$\mathcal{A}_0 \xrightarrow{a,s} \mathcal{A}_1$ with $a \notin \mathsf{Act}$. In this case we obtain $\varphi_1 = \varphi_0$ and $\psi_0 \| \mathcal{A}_0 \xrightarrow{a,s} \psi_0 \| \mathcal{A}_1$. So again we obtain the result by induction.*

Case 4: *Timed transition.....*

Now, since $(\varphi', \psi') \in R$ then $\mathsf{vio}(\varphi') \vdash \mathsf{vio}(\psi')$. But from break$(\mathcal{A}', \varphi')$ we can conclude that (i) $\mathcal{A}' \vdash \mathsf{vio}(\varphi')$. □

Proposition 4 *Let $\varphi, \varphi' \in \mathcal{C}$, then the following propositions hold:*

1. $\perp \preccurlyeq_\top \varphi \preccurlyeq_\top \top$
2. $\top \preccurlyeq_\perp \varphi \preccurlyeq_\perp \perp$
3. $\varphi \vee \varphi' \preccurlyeq_\perp \varphi$
4. $\varphi \wedge \varphi' \preccurlyeq_\perp \varphi$

Proposition 5 *Let $\varphi, \varphi', \psi, \psi' \in \mathcal{C}$, then the following propositions hold.*

If $\varphi \preccurlyeq_\bot \psi'$ and $\varphi \preccurlyeq_\bot \psi'$:	*If $\varphi \preccurlyeq_\top \psi'$ and $\varphi \preccurlyeq_\top \psi'$:*
$\bot.1$ $\varphi \blacktriangleright \psi \preccurlyeq_\bot \varphi \blacktriangleright \psi$	$\top.1$ $\varphi; \psi \preccurlyeq_\bot \varphi; \psi$
$\bot.2$ $\varphi \wedge \psi \preccurlyeq_\bot \varphi \wedge \psi$	$\top.2$ $\varphi \wedge \psi \preccurlyeq_\bot \varphi \wedge \psi$
$\bot.3$ $\varphi \vee \psi \preccurlyeq_\bot \varphi \vee \psi$	$\top.3$ $\varphi \vee \psi \preccurlyeq_\bot \varphi \vee \psi$

6 Runtime Verification of Timed Contracts

Although much work has been done on runtime verification [LS09] — 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 [CPS09, BLS06, BNF13]. 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. As we have previously done in [CLP17], we will be showing how we can automatically generate a runtime monitor from a contract, such that any violation of the contract is flagged.

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 [Brz64].

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 $e : es$ 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 $\psi \xrightarrow{a,k}; \longmapsto \psi'$ (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 $x\mathsf{wait}(t); \varphi$ would evolve to φ upon t time units elapsing. Similarly, if t time units elapse (with no system events received), we evolve the contract $\mathcal{O}_k(a)[t]; \varphi$ 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[1] $\varphi \overset{\text{timeout}(\varphi)}{\rightsquigarrow}; \longmapsto \varphi'$. Also, system events carry a timestamp, through which the contract can be moved ahead in time upon receiving the event.

The monitoring algorithm for our contract logic is shown in Algorithm 1. The state of the monitor is stored in variable *contract* and variable *systime* keeps track of the last timestamp processed by the system. Initially, these variables are set to ψ and 0. 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, the monitoring state is updated by first advancing time by $(t - \textit{systime})$ time units, and then stepping forward using the *step* function which returns the unique formula such that $\psi \xrightarrow{e}$ *step*(ψ, e) (lines 11–14). Finally, the *systime* variable is updated accordingly

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

(line 16).

```
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

7 Case Study

The case study presented in this section is inspired by an example described in [APS14]. It consists of a Plane Boarding System (PBS) involving two different agents: the passenger and the airline company. The following contract between the airline company and the passenger regulates their interaction during check-in and on the flight considering time constraints:

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 in 5 minutes. If she does not do so, she must return to the check-in desk in 15 minutes.

3. After presenting the boarding pass, the passenger must show her passport in 5 minutes.

4. Henceforth, the passenger is (i) prohibited from carrying liquids in her hand-luggage within 1 hour until boarding; and (ii) prohibited from carrying weapons during the whole trip until

the plane lands (t_1). 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 60 minutes and to carry hand-luggage within 10 minutes. The airline company is obliged to allow the passenger to board within 1 hour.

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.

In Table 9 we show the *PBS* contract by a list of the obligations, permissions and prohibitions that can be inferred from the description of the process.

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`)	2	5
2	*Passenger*	*Obligation*	Go back to the checkin desk (`GBCh`)	$\varnothing$	15
3	*Passenger*	*Obligation*	Show her passport (`ShP`)	$\varnothing$	5
4	*Passenger*	*Permission*	Board (`board`)	9 & 10	60
5	*Passenger*	*Permission*	Board with hand luggage (`hl`)	9 & 10	10
6	*Passenger*	*Prohibition*	Carry in her hand luggage weapons (`weapon`) until landing (`landing`)	$\varnothing$	60
7	*Passenger*	*Prohibition*	Carry in her hand luggage liquids (`liq`) until boarding (`board`)	8	60
8	*Passenger*	*Obligation*	Dispose of liquids (`dliq`)	$\varnothing$	10
9	*Airline Company*	*Obligation*	Put her hand luggage in the hold (`hlhold`)	$\varnothing$	20
10	*Airline Company*	*Obligation*	Allow passenger to board (`board`)	$\varnothing$	60

Table 9: Norms of the *Boarding System* contract

The contract described in natural language can be formalized using as follows:

$$
\begin{array}{rcl}
\varphi_0 & ::= & \mathcal{P}_p(\texttt{checkin})[t_0 - 120] \\
\varphi_1 & ::= & \mathcal{O}_p(\texttt{PBP})[5] \blacktriangleright \mathcal{O}_p(\texttt{GBCh})[15] \\
\varphi_2 & ::= & \mathcal{O}_p(\texttt{ShP})[5] \\
\varphi_3 & ::= & (\mathcal{F}_p(\texttt{weapon})[t_1]\mathcal{U}[\texttt{landing}, c]) \wedge ((\mathcal{F}_p(\texttt{liq})[60]\mathcal{U}[\texttt{boarding}, p]) \blacktriangleright \mathcal{O}_p(\texttt{dliq})[10]) \\
\varphi_4 & ::= & (\mathcal{P}_p(\texttt{board})[60]; \mathcal{P}_p(\texttt{hl})[10]) \blacktriangleright (\mathcal{O}_c(\texttt{board})[60];\ \mathcal{O}_c(\texttt{hlhold})[20]) \\
\textit{PBS} & ::= & \varphi_0; \varphi_1; \varphi_2; (\varphi_3 \wedge \varphi_4)
\end{array}
$$

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*.

8 Conclusions

In this paper we have presented a timed contract calculus, enabling us to reason about contracts, with time constraints, independent of the systems on which they are applied to. We have introduced a notion of bisimilarity between contracts, which takes into account predicates over system states. We have shown how the semantics can be used for the runtime verification of contracts, by translating contracts into the specification language of the LARVA runtime verification tool. Finally, we showed the utility of the timed calculus by applying it to a airline check-in desk case study.

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 perspectives, there are various questions we are 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.

References

[APS14] Shaun Azzopardi, Gordon J. Pace, and Fernando Schapachnik. Contract automata with reparations. In *Legal Knowledge and Information Systems - JURIX: The Twenty-Seventh Annual Conference, Jagiellonian University, Krakow, Poland, 10-12 December*, pages 49–54, 2014.

[APSS16] Shaun Azzopardi, Gordon J. Pace, Fernando Schapachnik, and Gerardo Schneider. Contract automata - an operational view of contracts between interactive parties. *Artif. Intell. Law*, 24(3):203–243, 2016.

[BDDM04] Jan M. Broersen, Frank Dignum, Virginia Dignum, and John-Jules Ch. 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*, pages 43–56, 2004.

[BLS06] Andreas Bauer, Martin Leucker, and Christian 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*, pages 260–272, 2006.

[BNF13] Borzoo Bonakdarpour, Samaneh Navabpour, and Sebastian Fischmeister. Time-triggered runtime verification. *Formal Methods in System Design*, 43(1):29–60, 2013.

[Brz64] Janusz A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, October 1964.

[CDMR01] James B. Cole, John Derrick, Zoran Milosevic, and Kerry 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 '01, pages 1–17, London, UK, UK, 2001. Springer-Verlag.

[CLP17] María-Emilia Cambronero, Luis Llana, and Gordon J. Pace. A calculus supporting contract reasoning and monitoring. *IEEE Access*, 5:6735–6745, 2017.

[CPS09] Christian Colombo, Gordon J. Pace, and Gerardo 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*, pages 103–117, 2009.

[DCMS14] Gregorio Díaz, M. Emilia Cambronero, Enrique Martínez, and Gerardo Schneider. Specification and verification of normativetexts using C-O diagrams. *IEEE Trans. Software Eng.*, 40(8):795–817, 2014.

[DS95] Jim Davies and Steve Schneider. A brief history of timed CSP. *Theor. Comput. Sci.*, 138(2):243–271, 1995.

[FPO+09] Stephen Fenech, Gordon J. Pace, Joseph C. Okika, Anders P. Ravn, and Gerardo Schneider. On the specification of full contracts. *Electr. Notes Theor. Comput. Sci.*, 253(1):39–55, 2009.

[GM05] G. Governatori and Z. Milosevic. Dealing with contract violations: formalism and domain specific language. In *EDOC Enterprise Computing Conference, Ninth IEEE International*, pages 46–57. IEEE Computer Society, 2005.

[LS09] Martin Leucker and Christian Schallhart. A brief account of runtime verification. *J. Log. Algebr. Program.*, 78(5):293–303, 2009.

[MM01] Olivera Marjanovic and Zoran Milosevic. Towards formal modeling of e-contracts. In *Proceedings of the 5th IEEE International Conference on Enterprise Distributed Object Computing*, EDOC, pages 59–68, Washington, DC, USA, 2001. IEEE Computer Society.

[New42] Maxwell Herman Alexander Newman. On theories with a combinatorial definition of" equivalence". *Annals of mathematics*, pages 223–243, 1942.

[PS09] Cristian Prisacariu and Gerardo Schneider. Cl: An action-based logic for reasoning about contracts. In *Logic, Language, Information and Computation, 16th International Workshop, WoLLIC, Tokyo, Japan, June 21-24. Proceedings*, pages 335–349, 2009.

[PS12] Gordon J. Pace and Fernando Schapachnik. Contracts for Interacting Two-Party Systems. In *FLACOS'12*, volume 94 of *ENTCS*, pages 21–30, 2012.

[Wyn06] Adam Zachary 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*, pages 255–271, 2006.

[Yi91] Wang 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*, pages 217–228, 1991.