Challenges in the Specification of Full Contracts *

Gordon J. Pace[1] and Gerardo Schneider[2]

[1] Department of Computer Science, University of Malta, Malta.
[2] Department of Informatics, University of Oslo, Oslo, Norway
gordon.pace@um.edu.mt, gerardo@ifi.uio.no

Abstract. The complete specification of full contracts — contracts which include tolerated exceptions, and which enable reasoning about the contracts themselves, can be achieved using a combination of temporal and deontic concepts. In this paper we discuss the challenges in combining deontic and other relevant logics, in particular focusing on operators for choice, obligations over sequences, contrary-to-duty obligations, and how internal and external decisions may be incorporated in an action-based language for specifying contracts. We provide different viable interpretations and approaches for the development of such a sound logic and outline challenges for the future.

1 Introduction

The specification and analysis of *full contracts*, i.e. contracts between different entities regulating not only the normal interactive behaviours but also the exceptional ones, is becoming an imperative in many Computer Science applications. These include service-oriented architectures, e-commerce, component-based software development, and any other application where there is a need for trustful collaborative interactions. Such contracts should express not merely the sequence and causality of events, but also what are the *obligations*, *permissions* and *prohibitions* of the participating entities. These three notions being the object of study of the so-called deontic logic.

The specification of such contracts needs a formal language rich enough to capture deontic notions, temporal and dynamic aspects, real-time issues (e.g. deadlines), and the handling of actions (events) and exception mechanisms. Besides, contracts must be amenable to formal analysis techniques (e.g. model checking and runtime verification). Furthermore, the use of contracts is only meaningful if there is a mechanism to monitor their fulfillment. We believe that such successful language should be built on top of a basic deontic logic, in combination with modalities and operators from temporal, dynamic, action-based, and real-time logics.

Deontic logic is the logic concerned with moral and normative notions such as obligation, permission, prohibition, optionality, power, indifference, immunity and intention, among others. Though the scope of the logic (from the philosophical point of view) is huge and there is no way to formalise all those notions and study their relation in one single formalism, it is usually recognised that a deontic logic must contain at least the notions of obligation, permission, and prohibition, and preserve their intuitive

* Partially supported by the Nordunet3 project "COSoDIS".

properties. Even when restricting the logic to just these three notions, its formalisation is not easy, as witnessed by the extensive research conducted by the deontic community both from the philosophical and the logical point of view (see [19,31]).

Besides obligations, permissions and prohibitions, contracts, agreements, and normative systems contain clauses which by definition may be violated, represented by *contrary-to-duty obligations* (CTD) and *contrary-to-prohibitions* (CTP). CTDs are statements that represent the fact that obligations might not be respected where CTPs are similar statements which deal with prohibitions that might be violated. Both constructions specify the obligation/prohibition to be fulfilled and which is the *reparation*/*penalty* to be applied in case of violation.

We believe that when restricted to specific domains, deontic logic is a practical powerful specification tool, if combined with the above-mentioned logics. However, it is well known that deontic logic is not useful unless we ensure the absence of paradoxes and practical oddities [19,25]. Many of these paradoxes are due to the problematic combination of classical propositional logic operators with sequences, choices, repetitive behaviour, and CTDs and CTPs.

In this paper, we outline some of the choices to be made, and the challenges to be faced when combining deontic notions with other useful temporal concepts in the definition of a formal language for full contracts. Frequently, contracts are perceived simply as properties to be satisfied by a system. However, the analysis of the contracts as first class entities along the lines we are presenting it in this paper, can be fruitful in a variety of contexts especially in situations where services may require to be composed. Each service may come with its own contract, and the analysis of the composition of the constituent contracts may be necessary to study interaction of such services. One may also want to make queries about the contract, for instance to know what are the obligations of each participant and the penalties in case of not fulfillment of those primary obligations. Moreover, the decisions taken in the design of such a language are crucial since the ultimate goal is to have a framework facilitating contract analysis, as for instance: (i) Check that a particular system satisfies a contract; (ii) Reason about the contract itself directly; (iii) Reason about the relation between different contracts.

The paper is organised as follows. In the next section we argue for a ”practical” deontic logic, base for a formal contract language, highlighting general aspects. In section 3 we present our language of discourse for specifying full contracts. In the following sections we discuss the problems arising with the formalisation of obligation, CTDs, sequences, regular expressions and internal vs. external operators, in combination with other concepts from temporal, dynamic and real-time logics.

2 What is needed for a “practical” deontic logic

As presented in the previous section, we claim that a good language for the specification of full contracts needs a practical deontic logic. The use of deontic logic goes beyond the standard formalisation of legal contracts, abstracting from other “modal-

ities" and subtleties[3] and concentrating on obligations, permissions and prohibitions and their relation. Our aim is to restrict deontic logic to avoid paradoxes, and extend it accordingly as to be useful in the following contexts: (1) Fault-tolerant systems; (2) Compensable transactions; (3) Regulatory systems (4) Service-oriented architectures; (5) Component-based development.

Both fault-tolerant systems and systems with long transactions have in common that the specified desirable (*mandatory*) behaviour (sequence of states) will not necessarily be respected due to failures. In the presence of such failures, for some specified reason, we want to be able to come back to a previous state where an alternative behaviour must be enforced. This is very much what a CTD (or CTP) means. In some other cases we rather want to describe this deviation from expected mandatory behaviours as exceptions. Regulatory systems are normative systems containing regulations, laws and policies, rich on clauses specifying not only primary obligations but also exceptions. Such documents abound on cross-references to other clauses intra- and inter-document. In the context of SOA a deontic based approach may serve the purpose to give semantics to Service Level Agreements, or SLAs (which currently lack formal semantics) or to be used just as a specification language to write contracts between services. In component-based software development, contracts may be attached to components in order to guarantee, among other things, their compatibility both at development and deployment time.

Due to lack of space we will only indicate few papers where examples can be found. See for instance [26] for an example on the formalisation of a legal contract, and [10,7] for a justification on the use of deontic logic, and examples, on fault-tolerant systems. See [14,16] for compensable transactions, [11] for regulatory conformance checking, and [24] for state of the art and challenges in SOA. See [23] for a discussion of the use of contracts in the setting of component-based systems.

In order to avoid an inconvenient generalisation on the use of the term "deontic logic" we will restrict ourselves in what follows to a variant of deontic logic, which we believe is expressive enough to handle the representative number of applications mentioned above. We will call *OPP-logic* a logic or formal language containing the deontic modalities of Obligation, Permission and Prohibition (OPP), defined over complex actions, obtained from basic actions by (a restricted) combination of the following operators; choice $+$, sequential composition $\cdot$, concurrency $\&$, Kleene star $*$, and action negation $\bar{\cdot}$. Moreover, in OPP-logic it should be possible to specify: (i) Nested CTDs and CTPs; (ii) Temporal (causal) aspects; (iii) Nested Exceptions; (iv) Real-time aspects; (v) References to other expressions or clauses; (vi) Invariants; (vii) Fairness constraints; (viii) Contract introspection/reflection. Due to lack of space we will mainly concentrate on the first three aspects above, though we will also comment on the other items without entering into details.

[3] One may want to make a distinction between "rights" and "permissions", between having the power to do something and the permission to do so, or even between "must", "ought to" and "should", but for certain kind of analysis these distinctions are not needed.

3 The Language of Discourse

In this section, we outline the syntax of a language for the purpose of expressing full contracts. Please note that the language presented is not intended to be a complete formal language, but rather a language rich enough for us to illustrate issues in the use of a logic in expressing contracts over systems, in the spirit of the OPP-logic discussed above. The semantic issues will be discussed informally in later sections of the paper.

Actions An important decision is that of whether the deontic operators act upon the state or the actions influencing the state. The two views, already familiar to computer scientists in the domain of specification languages[4], can both be defended, depending on the domain of application. We will base our approach on an action-based logic. As in the domain of process calculi, we go beyond simple actions to include parameterised actions (for example, *pay* may be a parameterised action whose parameter specifies the amount paid), and collections, or multisets of actions (to enable concurrent actions, including multiple instances of the same action). We also will use the notation $\overline{act}$ for any action different from act.

$$\text{Action} ::= \varepsilon \mid \text{Any} \mid \text{SimpAction} \mid \text{SimpAction(Param)} \mid \text{Action} \ \& \ \text{Action} \mid \overline{\text{Action}}$$

We will use lower-case Latin letters, $a, b, c, \ldots$ to denote basic actions.

Expressions over actions Since we are interested in behaviour over time, to reason about causality, we require the enriching of actions over a temporal logic with operators such as sequentiality, choice and repetition. For the purposes of this paper, we will use extended regular expressions over actions, with + for choice, * for repetition, . for sequencing, & for concurrency and $\neg$ for negation.

$$\begin{aligned}\text{CompAction} ::= \ & \text{Action} \mid \neg \ \text{CompAction} \mid \text{CompAction}^* \mid \text{CompAction} \ + \ \text{CompAction} \\ \mid \ & \text{CompAction} \ \& \ \text{CompAction} \mid \text{CompAction} \ . \ \text{CompAction}\end{aligned}$$

We will use lower-case Greek letters, $\alpha, \beta, \ldots$ to denote compound actions.

Deontic operators We will be using three basic deontic operators: permission, prohibition and obligation, which can be applied on compound actions. The basic contracts $\mathbb{Y}$ and $\mathbb{N}$, respectively corresponding to the trivially satisfied and unsatisfiable contracts, will also be included for completeness.

$$\text{SimpContract} ::= \mathbb{Y} \mid \mathbb{N} \mid \mathbb{P}(\text{CompAction}) \mid \mathbb{F}(\text{CompAction}) \mid \mathbb{O}(\text{CompAction})$$

Default contracts Rather often, contracts are composed of a cascade of contracts: you are obliged to do something, but if you do not, you are then obliged to do something else. In general, one may define a defaulting operator over contracts which, given two contracts, behaves like the first, but enacts the second if the first is broken. However, we will limit ourselves to two simple forms of this operator in this paper: to Contrary-To-Duty (CTD) contracts, and Contrary-To-Prohibition (CTP) contracts. A CTD is made up of a compound action, and another contract — the performance of the action is obliged, but if not performed, the contract is enacted.

[4] Specification languages such as Z, enable a state-based view of the system, as opposed to process calculi such as CSP and CCS, where the emphasis is on the actions.

Similary CTPs enact a prohibition by default. In other cases we may rather consider exceptions, where a given contract is not enforced if another one is fulfilled.

$$\begin{aligned}\text{CompContract} ::= {} & \text{SimpContract} \mid \text{CTD(CompAction, CompContract)} \mid \\ & \text{CTP(CompAction, CompContract)} \mid \\ & \text{CompContract } unless \text{ CompContract}\end{aligned}$$

Expressions over contracts We also require temporal operators over contracts (for instance, to be able to express that an obligation is enacted as soon as another obligation is satisfied). We will similary use regular expressions for this purpose. We add another operator to test for the presence of an action, with $a?.C$ being equivalent to contract C (as of the next time unit) if a is present now, but void (for which nothing needs to be done to satisfy it) otherwise.

$$\begin{aligned}\text{Contract} ::= {} & \text{CompContract} \mid \neg\text{Contract} \mid \text{Contract}^* \mid \text{Contract } + \text{ Contract} \mid \\ & \text{Contract } \& \text{ Contract} \mid \text{Contract . Contract} \mid \text{CompAction? . Contract}\end{aligned}$$

The use of contract negation may seem an odd choice — the interpretation of negation is that the particular contract is not enacted. For example, one can express permission to perform an action, as the negation of the obligation to perform something other than the action. We will not provide a formal semantics of the language since our intention is to explore the design and problems of such formal language and not its particular formalisation.

4 Combining Temporal and Deontic Notions

4.1 Sequences

We need sequences of actions if we want to distinguish between situations such as "You are obliged to fill in the form, and then sign it" from "You are obliged to fill in the form, after which you are obliged to sign it".

Sequences over Contracts, and Contracts over Sequences The choice of including sequences both inside and outside contracts is arguably necessary for a number of reasons. The semantic difference between a statement such as $\mathbb{F}(a.b)$ and $\mathbb{F}(a).\mathbb{F}(b)$ is rather straightforward — while the former disallows the *sequence* of a followed by b, the latter forbids a, after which it forbids b. While the former allows an action a, followed by c, the latter is broken upon the arrival of action a. The distinction is similarly clear with permissions. However, the distinction when it comes to obligations of sequences is finer [32]. In DDL (Dynamic Deontic Logic) sequences of obligations (SoO) and obligations on sequences (OoS) are equivalent [21], though they should not be considered equivalent [32]. This is indeed clear if we consider what is the consequence of fulfillment and violation of such contracts. In SoO one can associate different reparations to each element on the sequence, while this is not the case in OoS (see [32] for examples). The need for distinguishing between sequences of contracts and contracts over sequences is however justified on a number of criteria.

Elegance of expression: It can be argued that contracts over sequences can be encoded as sequences over contracts. For instance, $\mathbb{F}(a.b)$ can be (up to a certain extent) be encoded as $a?.\mathbb{F}(b)$, similarly $\mathbb{O}(a.b)$ could be written as $\mathbb{O}(a) \wedge a?.\mathbb{O}(b)$. If a contract's

semantics cares only about the instances of failure (breaking the contract), the contracts could be argued to be similar. However, the rewritten contracts (as sequences of contracts) lose the direct intuitive meaning originally meant and expressed as contract over sequences. Furthermore, without introducing a general contract default operator, expressing CTDs and CTPs over contracts on sequences introduces further complications — for example, $\text{CTD}(a.b, C)$ could be rewritten into something of the form $a?.\text{CTD}(b, C) \wedge \bar{a}?.C$.

Contract violation: If a prohibition clause over a sequence $a.b$ is expressed using a conditional term $a?.\mathbb{F}(b)$, the first action a is not seen as an explicit part of the prohibition. The interpretation of such a term indicates that breaking the contract results from performing action b which is forbidden after an action a. This is in direct contrast with the intended interpretation which forbids the sequence $a.b$.

Reasoning about contracts: Furthermore, the failure semantics of a contract is but one interpretation. Other views of the contract may include analysis directly related to the deontic operators in a contract (such as which contracts are active at a particular point in time, the total number of obligations in a particular contract, etc). Furthermore, if one extends the deontic language to refer to use the presence or absence of obligations and prohibitions as conditions within the language, action sequences become indispensable.

Causality Another interesting issue is that of causality. Consider the semantics of the default contract operators such as $\text{CTD}(\alpha, C)$. The informal interpretation of the operator is that an obligation to perform α is enacted, but if it is not, contract C has to be satisfied. However, this leads to two differing views of the operator: (i) contract C must hold as soon as (or one time unit after) the initial obligation is broken; or (ii) the choice between performing the obligations or the alternative contract C as soon as the CTD is enacted. In the literature, the first interpretation is typically given. The second interpretation leads to interesting causality situations. Consider the contract $\text{CTD}(\text{Any}.a, \mathbb{O}(b))$ — if one views $\mathbb{O}(b)$ as an exception/option, it has to be done before breaking the original obligation. An initial action b may satisfy the CTD or not — there is no way we can know this until we get the second set of events.

Breaking an obligation CTDs and contracts with exceptions still prove to be challenging with sequences of actions. Consider the following situation:

> *The law of a country says that: 'You are obliged to hand in Form A on Monday and Form B on Tuesday, unless officials stop you from doing so.'*
>
> *On Monday, John spent a day on the beach, thus not handing in Form A. On Tuesday at 00:00 he was arrested, and broght to justice on Wednesday.*
>
> *The police argue: 'To satisfy his obligation the defendant had to hand in Form A on Monday, which he did not. Hence he should be found guilty.'*
>
> *But John's lawyer argues back: 'But to satisfy the obligation the defendant had to hand in Form B on Tuesday, which he was stopped from doing by officials. He is hence innocent.'*

Who is right? Formalising the primary obligation in the law, we get $\mathbb{O}(a.b)$, where a represents handling Form A on Monday and b handling Form B on Tuesday. When is the obligation to be considered violated — upon the lack of action a, or at the end of two consecutive actions? It will depend on whether we model the above with CTDs or "unless", and what is the formal semantics. To avoid this and similar paradoxes, we propose the use of an earliest failure semantics, in which, a contract fails as soon as it can no longer be satisfiable.

CTDs and Sequences of Actions The introduction of time into the deontic soup raises the question of what are the most natural semantics to CTDs. For example, let us consider CTD$(a, \mathbb{O}(b))$. Does this correspond to an obligation to do a, which *if violated*, will then set up an obligation to perform a b, or can the b be performed immediately to satisfy the contract. In other words, does the sequence of actions $(\bar{a}\&\bar{b}).b$ satisfy the contract? What about $\bar{a}\&b.\bar{a}$?

The enactment of the compensation contract (and hence the possibility of satisfying it) only after the first obligation is violated distinguishes CTDs from mere choice, even if external choice over contracts may be desirable in certain contexts. This will be further discussed in the coming sections.

Introducing sequences of actions into a CTD is directly related to the paradox given above. If the moment of violation of the obligation is used as the triggering of the new (compensation) contract, earliest failure semantics may be necessary.

4.2 Choice

The choice operator has an intuitively different semantics when given inside, or outside a deontic operator. The contract obliging you to hand in Form A or Form B, is distinct from the disjunctive contract which says that either you are obliged to hand in Form A, or you are obliged to hand in Form B. The interpretation of the latter contract has been much discussed in the literature, and various paradoxes are known, which challenge various naïve semantics of the operator. In our opinion, it is not advisable to have the classical disjunction in contracts, or at least to restrict its use, since De Morgan rules may introduce paradoxes.

Choice of Obligations, and Obligations of Choices Let us start by considering the contract with an obligation over a choice: $\mathbb{O}(a + b)$. As in standard process calculi, the choice operator can have (at least) two radically different views: angelic vs demonic, or sometimes internal vs external choice. Does performing action a always satisfy the contract, or is the choice made internally (demonically) and you may be obliged to perform action b? Similarly, the choice over contracts has a similar issue — $\mathbb{O}(a) + \mathbb{O}(b)$. The distinction between the two as used in natural language contracts can be made clearer by considering financial contracts which John signs with Peter:

Contract 1: 'On the 1st of May, John will either (i) be obliged to sell 100 shares at \$1 each; or (ii) be obliged to sell 50 shares at the market price.'

Contract 2: 'On the 1st of May, John will be obliged either (i) to sell 100 shares at \$1 each; or (ii) to sell 50 shares at the market price.'

The (possibly debatable) interpretation of the contracts is that, while in contract 1 the choice of which obligation to enact lies with Peter, in the latter one obligation is enacted, and it is up to John to decide how to discharge it. Peter should prefer the first contract, whereas John should prefer the second.

This interpretation corresponds to having choice outside the deontic operators to be resolved by the contract controller (demonic, or internal choice), while choice inside deontic operators to be resolved at the entity upon whom the contract acts (angelic, or external choice).[5] The only way to, *a priori*, guarantee satisfying the contract

[5] Note that here internal and external refers to the contract, and not to the performer of the action.

$\mathbb{O}(a) + \mathbb{O}(b)$ is through performing both actions a and b, which may be impossible if the semantics of $+$ is given as an exclusive or.

This raises the question, once again, whether the logic, or implementation should have access to currently enacted obligations. In such cases, one would be able to write something on the lines of $O_a?.a + O_b?.b$ ($O_a?$ says whether an obligation to do a is active right now, continuing without any delay in time) to guarantee that the contract $\mathbb{O}(a) + \mathbb{O}(b)$ is always satisfied. On the other hand, having access to this information within the logic may lead to contructive anomalies with contracts such as $\neg O_a?.\mathbb{O}(a)$. Constructiveness of such logics has been studied in other contexts such as synchronous programming [28] and cycles in circuits [9].

Let us consider now the contract: You are forbidden from doing a or b, written as $\mathbb{F}(a + b)$. In $\mathbb{O}(a + b)$ we seem to take it to mean 'you must do something which creates a trace satisfying the regexp $a + b$'. In $\mathbb{F}(a + b)$, we are saying 'you must not do something which creates a trace satisfying the regexp $a + b$'. What is the meaning of $+$ in $\mathbb{F}(a + b)$? It seems like the choice inside a forbidden operator becomes an internal choice, not an external one. The implicit negation outside the $\mathbb{F}$ switches the $+$ from external to internal. Is this a suitable interpretation? It depends. If we define $\mathbb{F}(a + b)$ to be $(\mathbb{F}(a) \wedge \mathbb{P}(b)) + (\mathbb{F}(b) \wedge \mathbb{P}(a))$ it seems that the above interpretation is correct. On the other hand, we have a different interpretation if we consider that $\mathbb{F}(a + b)$ to be defined as $\neg\mathbb{P}(a + b)$, where $\mathbb{P}(a + b)$ means the same as $\mathbb{P}(a) \wedge \mathbb{P}(b)$, in which case we get that both a and b are forbidden.

Clearly, it is debatable whether external and internal choice should be separate operators or a single operator with different interpretations in different modalities.

The Moment of Choice and the Moment of Contract Satisfaction Differentiating between internal and external choice makes it possible to express different contracts in a natural way. However, another parameter which has to be decided for a temporal deontic logic is *when* the choice is made. Consider the non-deterministic contract Any?.$\mathbb{O}(a)+$ Any?.$\mathbb{O}(b)$. In this example, Any? acts like the silent action τ in process calculi such as CCS [22], and raises the question of whether this contract is in any way different from Any?.$(\mathbb{O}(a) + \mathbb{O}(b))$. The choice between contracts can be made immediately, or later on in the future, as soon as a contract is to be enacted. Therefore, to write a contract which leads to different obligations depending on the presence (or otherwise) of a particular action, one would express it as: $(a?.\mathbb{O}(b)) + (\bar{a}?.\mathbb{O}(c))$.

On the other hand, due to the user-centric view of angelic choice, the satisfaction of a contract with choice may yield different interpretations. Consider a contract such as $\mathbb{O}(a + a.c).\mathbb{O}(d)$. After receiving an action a, we are unsure whether the first contract has been satisfied, since it depends on whether the user will proceed with $c.d$, or simply with d. Similarly, given the contract $\mathbb{O}(a+a.c).\mathbb{O}(c)$, it is non-deterministic whether the action sequence $a.c.c$ satisfied the contract after the first two, or three symbols. Besides the above technicalities concerning non-determinism, we argue for forcing deterministic contracts, as desirable both in the legal arena as in our mentioned applications.

Choice and CTDs If choice between contracts leads to internal, nondeterministic choice, a contract such as CTD$(a, b)+$CTD(c, d), may be broken if one performs an action a but no c (and no d to compensate). Without having access to how the choice over contracts is resolved, leads to having to satisfy both contracts. With CTDs, such a composition

of contracts may lead to unsatisfiable contracts in roundabout, counter-intuitive ways. Furthermore, the issue of when a contract over sequences is violated rises again, since the CTD is triggered upon violation.

Note that the choice operator may be seen as a kind of exclusive disjunction. However, this similarity is only apparent as clear in the case of writing the following contract: $\mathsf{CTD}(a, \mathbb{O}(b)) + \mathsf{CTD}(b, \mathbb{O}(a))$. With an exclusive or point of view, the above will lead to a violation independently of what is done, as performing a may be seen as satisfying the primary obligation in the first disjunct, but also the reparation in the second; similarly with b. The above example is problematic even under the interpretation of $+$ as choice, if non-determinism is allowed.

4.3 Repetition

The use of repetition in contracts, corresponding to a combination of choice, sequences and fix-points, poses a variety of challenges related to the ones already discussed. As in the case of choice and sequences, a contract stating that 'John is repeatedly obliged to pay after which he is permitted to use the service', is different from 'John is obliged to pay repeatedly after which he will be permitted to use the service' — $\mathbb{O}(p)^*.\mathbb{P}(s)$ as opposed to $\mathbb{O}(p^*).\mathbb{P}(s)$.

Notation Different approaches to logic use different operators for repetition. Languages such as $\mathcal{CL}$ [26] use the LTL- and CTL-style until operator. In this paper, we use the regular expression-style star operator to indicate repetitions. This gives a uniform view of temporal operators inside, and outside the deontic operators, and enables repetition of contracts which take more than one time unit to terminate. For instance, the contract $\mathbb{O}(5c.10c + 50c)^*.\mathbb{P}(choc)$ will repeatedly obliges inserting 5 and 10 cent coins in sequence or a 50 cent coin, until at the end withdrawing a chocolate is permitted. Similarly, one can give a contract $\mathbb{O}((5c.10c+50c)^*).\mathbb{P}(choc)$, which has a single obligation for the user to insert any number of coins, after which she is permitted to withdraw a chocolate. Should the two contracts be distinguishable?

Expressing contracts with action sequences within the deontic operators using an until operator can prove to be challenging. It is certainly desirable to have the inner temporal logic match (at least in style) with the outer one. Using an interval logic (regular expressions) inside the deontic operators and a point logic outside the operators can result in spin the presence of CTDs (and CTPs).

Contracts of Repetitions and Repetitions of Contracts Informally, equating the star operator with an unbounded sequence of choices indicates that the interpretation of the choice operator inside and outside the deontic operators should be respected with repetition. In other words, a contract such as $\mathbb{O}(a^*)$ (intuitively equivalent to $\mathbb{O}(\varepsilon + a + a.a + \ldots)$) means that a number of actions a are to be performed — the choice regarding the number of repetitions is external, that is, decided by the entity bound by the contract. On the other hand, with $\mathbb{O}(a)^*$ (intuitively equivalent to $\mathbb{Y} + \mathbb{O}(a) + \mathbb{O}(a).\mathbb{O}(a) + \ldots$), the choice regarding the number of repetitions is internal, and thus imposed.

Unbounded Repetition Let us consider the contract saying that 'If John uses the service, then he is bound to eventually pay.' One would write this as $s?.\mathbb{O}(\mathrm{Any}^*.p)$. Note

that no bound is placed on how long John takes to pay his dues. Giving a formal semantics of the logic over infinite sequences enables one to say whether or not John has satisfied the contract. On the other hand, when one looks at finite sequences, one requires the use of a three-valued logic to differentiate between the contract being violated, satisfied, and the third situation when it may still be satisfied in the future. In certain contexts, such as runtime verification, this approach will be required. In practice, however, it seems more natural to have only bounded iteration, to be able to enforce the payment in a life-time period. However, one may question whether unbounded repetition *inside* deontic operators is meaningful. John would certainly have no qualms about signing the above-mentioned contract, since he is not due to perform any action on his part to satisfy the contract. On the other hand, unbounded repetition outside deontic operators is still meaningful, given that the choice over repetition is an internal one.

4.4 Other Issues

Real-Time Aspects Most useful contracts include some timing aspects: deadlines, timeouts, durations, etc. Dealing with real-time introduces further challenges when combined with deontic notions: Should we associate time with the modalities, clauses, actions, or with all of them? Is an interval-based logic necessary to reason about the beginning and end of an action?

Since we want not only to specify but also to be able to analyse contracts written in our logic, we aim at a decidable extension. Though many of the above decisions may be application-driven, we propose to consider the use of *clocks* with freezing quantifiers and resets [1]. A modular conservative approach would be to extend an untimed OPP-logic with clocks. In this way a suitable combination of Kripke-like structures with timed automata could be envisaged as a semantical framework for such a logic.

Reference to other expressions A *nominal* logic or simply annotations on clauses and contracts may be needed to be able to refer to other clauses (in the same contract) or contracts. In principle the analysis of cross-references could be analysed with standard existing techniques on graph (e.g. reachability analysis).

Introspection / reflection One interesting extension is that of introducing contract introspection — the capability of having conditions which depend on which obligations, permissions and prohibitions are active, to express contracts such as 'Whenever you are obliged to pay, you are also obliged to produce identification'. Furthermore, a contract may contain references to itself, i.e. be *reflexive*, for instance a clause may determine that under certain circumstances a party may have the *power* to change other clauses, or even to cancel the contract. Note that, not only does this complicate compositional analysis of contracts, but may also introduce causality issues. Related to contract introspection, we could also envisage to have a formal theory to relate *policies* and contracts (i.e. "vertical" contracts regulating "horizontal" contracts).

Fairness and invariants Other issue is how easy is to represent *fairness* and *invariants*. In what concerns invariants, we may be able to represent the "box" operator of LTL in a similar way as it is defined in Duration Calculus, i.e. by negating a particular contract sequence. Concerning fairness, we want to specify properties like "any infinitely often enabled process should be infinitely often taken" (or other variations of fairness

constraints usually defined in temporal logics). As in temporal logics, we may decide to take a syntactic approach (e.g. as in LTL) in which case the logic should allow writing something like $\Box\Diamond C$, or a semantic approach (e.g. *à la* CTL). Again, in practice it seems reasonable to enforce only bounded fairness.

Concurrency In many applications it seems natural to consider true concurrency, mainly under the deontic operators. For instance, one may need to say that 'you are obliged to sit-down and remain silent', $\mathbb{O}(s\&r)$, where an eventual violation of this obligation includes not doing any (nor both) actions. The combination of such concurrent operator introduces many challenges and its combination with the other regular expression operators is not easy. One problem is that depending on the interpretation of conjunction on obligations, it may be difficult to assert whether $\mathbb{O}(s) \wedge \mathbb{O}(r)$ entails $\mathbb{O}(s\&r)$, or whether they are equivalent. In the latter this introduces a big challenge concerning the semantic interpretation of $\&$ in a deontic logic based on actions in the style of dynamic logic, since conjunction is usually interpreted as a branching in certain cases. See [27] for further discussion on the topic.

Conditional Contracts The use of conditions outside the deontic operators enables the formulation of contracts which are dependant on runtime behaviour. The question of whether such conditions should also be allowed inside the deontic operators leads to interesting possibilities. Consider the contract 'Unless the service is disabled, John is obliged to pay in the next time unit' as opposed to the contract 'John is obliged to pay in the next time unit, unless the service is disabled now.' The former obligation is never enacted if the service is disabled, whereas the latter is enacted, but becomes trivially satisfied when disabling the service. The former corresponds to $\bar{d}?.\mathbb{O}(p)$, while the latter to $\mathbb{O}(\bar{d}.p + d)$ (or even Any.$\mathbb{O}(p)$ $unless$ $d?$). Different approaches can be used to express such conditional contracts.

Encoding as normal actions: As seen in the example, one can encode conditional contracts using regular expressions. The main drawback of this approach is that which actions are conditions, and which are actually obliged by the contract becomes blurred. Although for most uses of a contract such a view is sufficient, when analysing a contract (for instance, to identify what actions one may be obliged to take) one may require further information.

Subjects, objects and actors: One approach is the identification of the actors of a contract — who the subject of an action is, and who the object is. Automatically, actions which are not performed by the party being obliged to do something, are conditions. However, this approach fails when the condition includes actions under the control of the party. In the above example, John may be the actor who chooses whether or not to disable the service. Introducing the subject and object of an action gives insight into a contract, but not sufficient information to analyse conditions.

Assertions: Actions (or regular expressions over actions) may be explicitely tagged as conditions, or assertions to indicate that they are not part of the obligation or prohibition: $\mathbb{O}(d? + \bar{d}?.p)$. The semantics to such a statement, is that if d is not present, then p must follow, otherwise the obligation becomes trivially violated. Although one may also give a semantics in which the default is the trivially satisfied contract, this leads to confusing situations — consider the contract 'John is obliged to pay until he disables the service'. This can be written as $\mathbb{O}((\bar{d}?.p)^* d?)$, which intuitively corresponds

to $\mathbb{O}(d? + \bar{d}?.p.d? + \bar{d}?.p.\bar{d}?.p.d? + \ldots)$. Recall that we suggested that a logical choice for the semantics of choice inside deontic operators is an external (user-driven) choice. But by not disabling the service and choosing the first choice in the unrolled version of the contract, the contract is trivially satisfied. Rewriting the contract to work with such semantics would require an implicit choice operator within deontic operators, or use unnecessarily intricate formulations such as: $\mathbb{O}((d?.0 + \bar{d}.p)^*(d? + \bar{d}.0))$ (where 0 corresponds to an action that can never be performed).

Another choice in the design of conditions in a logic for contracts is that of whether they should take time to evaluate. If the original contract discussed in this section were 'John is obliged to pay, unless the service is disabled,' one would have to reformulate the formal contract using conjunction $\mathbb{O}((\bar{d}?\&p) + d?)$. An alternative is to have conditions (both inside and outside deontic operators) with no time to evaluate, and proceed in the same time unit. Having such an approach makes expressing certain properties more straightforward, but at the cost of the need for constructiveness checking to avoid expressing counter-intuitive, or meaningless properties such as $\bar{d}?.d$ (if d is not present in the current time unit, then ensure it is).

5 Final Discussion

We have argued here that a formal language for full contracts needs a 'practical' deontic logic in combination with features from many other logics, and we have discussed some problems in obtaining such a logic. Giving a sound formal semantics to a real-time temporal deontic logic is not an easy task. Different approaches may be necessary for different application areas. To be able to discuss CTDs and CTPs (or default contracts, in general), a failure-oriented semantics is very desirable. Equivalence of contracts is another major challenge, and a bisimulation based approach can prove to be fruitful — the main question is to define reasonable bisimulation relations to encompass the different views of what equivalence over contracts means. Analysis of contracts is another important area worthy of investigation. We have recently developed a model-checking technique for the discovery of contradictory contracts in $\mathcal{CL}$ [13]. However, other analysis techniques are required — constructiveness analysis ensures no causality cycles in a contract, timing analysis may be used to study the lifetime of a contract, etc.

In identifying a number of challenges in combining temporal and deontic operators, we have mentioned various extensions which introduce more expressivity, but also further complexity into the logic. These include introspection, real-time issues, cross-references, a suitable treatment of true concurrency, fairness and conditional contracts

Also, since we have considered an *ought-to-do* approach (i.e. the deontic modalities are applied on actions and not state-of-affairs), one may also have to consider many aspects from dynamic logic, in the spirit of Meyer's work [21] (see also [4,26]).

Finally, we have used a simple temporal logic (regular expressions) onto which to graft our deontic logic. One may choose to start from another logic such as LTL, CTL or μ-calculus. The general question of how one can uniformly extend a (discrete) temporal logic with deontic operators is an interesting one, giving a unified view of a *deontic transformer* (in the style of monad transformers [17] in functional programming).

Besides specific technical differences with problems identified in previous work on deontic logic (see related work below), our work is set apart by the fact that some challenges in the specification of full contracts go beyond the "traditional" research by the

deontic community. Thus, the definition of a formal language (and reasoning system) for full contracts in Computer Science applications, involves additional challenges, as for instance: (1) Combination with other logics, in particular real-time logics; (2) The need of a *trace semantics* for the contract language, since contracts must be monitored at runtime; (3) An algorithm to automatically obtain such a runtime monitor; (4) Eventual use of an enforcement mechanism, at runtime; (5) Development of a *contract-as-types* theory in order to manipulate contracts at the programming language level.

Related Work Puzzles and paradoxes have accompanied deontic logic since its very beginning [30], starting with its formalisation in the so-called standard deontic logic (SDL). See von Wright's account [31] and McNamara's article [19]

Problems in formalising CTDs in the original papers on deontic logic were first discussed by Chisholm in [8]. Åqvist [2] gave a solution to the paradoxes related to CTDs (and also to the Good Samaritan and the paradox of the epistemic obligation) by proposing different semantic relations for primary and reparational obligations. The problem with the proposed solution is that one might need an unbounded (and eventually infinite) number of such relations in case of an unbounded number of nested CTDs. Prakken and Sergot [25] further discussed the difficulties to get a good representation of CTDs, based on an ought-to-be approach. In particular they showed how non-monotonic methods are not suitable for such cases, since such logics (e.g. *defeasible* logic) treat CTDs as exceptions, in which case the primary obligation is not really violated, since it is never applied. The difference between a CTD and an exception may be seen in the following example: Let A be 'you are obliged to pay unless somebody else pays for you', and B be 'you are obliged to pay, and if you do not pay somebody else has to pay for you'. A expresses an exception where 'somebody else pays for you' can be seen as a rule *defeating* the first obligation, which is never enacted in case the exception happens. However, in B, whenever you do not pay and somebody else does it for you, there has been a violation of the primary obligation, and the fact that somebody pays for you is seen as a reparation to the violated obligation. The difference is not naïve, and contracts A and B should be seen as different.

The problem of sequence of obligations versus obligation on sequences, and their combinations with CTDs has been discussed by Wyner in [32]. Wyner argues that none of previous work (including [6,21]) have given a good unproblematic representation of CTDs, in particular its relation with sequences and obligations. Khosla and Maibaum [15] mentioned these differences but did not propose a suitable solution. Wyner further studies the distinction, making a difference between 'distributed obligations´ and 'obligations on an interruptible sequence', corresponding to our proposal of external and internal sequences on obligations.

A discussion on the difference between internal and external choice in deontic logic has been addressed for instance in [20] and more recently in [29].

In [18] Lomuscio and Sergot describe a way to distinguish between normal and exceptional cases in their deontic interpreted systems, from the semantical point of view, by separating "allowed" from "disallowed" states.

A different approach based on dynamic logic by Meyer [21], where a special *marker* V for violation is introduced in order to mark that an obligation has been violated (or a forbidden action has been performed) in the current world (where V holds). Though the approach makes a big step towards a solution of many of the deontic paradoxes, it

does not properly address the problem of what follows after a violation. Furthermore, in Meyer's logic it is a theorem that sequences of obligations are equivalent to obligation on sequences, which is not the case as we have argued in section 4.1.

An example of the use of a restricted deontic logic for fault-tolerant systems is presented by Castro and Maibaum [7] (see also Coenen's work [10]). As far as we know, there is no application of deontic logic to systems with compensable transactions. Most of the research conducted in this domain uses process calculi (see for instance [16,5]).

A choice operator explicitly appears in those works using the ought-to-do approach, where the operator is among actions ($a + b$), for instance in the works by Broersen et al [4], and Prisacariu and Schneider [26]. The latter also allowing concurrent actions.

A logic with deontic flavor has been recently introduced by Dinesh et al. in [11,12]. The main objective of that work is to represent conditional obligations and permissions, to capture exceptions to norms, and most notably to be able to represent (and reason) about references among norms (clauses). As far as we know, it is not possible to represent CTDs and CTPs in that logic. However, the application domain of the paper is the regulatory system of blood donations, where CTDs and CTPs are not common.

The recent paper by Åqvist [3] proposes a formalisation of a logic for conditional obligations and permissions, combined with temporal modalities. No CTDs, CTPs, nor real-time is considered.

Finally, note that the concept of *contract* used in this paper is more general than behavioural interfaces, the design-by-contract programming style using pre- and post-conditions, and other more "standard" notion of contracts in Computer Science (see [23] for a discussion on that).

References

1. R. Alur and T. A. Henzinger. Real-time logics: complexity and expressiveness. *Information and Computation*, 104:390–401, 1993.
2. L. Åqvist. Good samaritans, contrary-to-duty imperatives, and epistemic obligations. *Noûs*, 1(4):361–379, 1967.
3. L. Åqvist. Combinations of tense and deontic modality: On the approach to temporal logic with historical necessity and conditional obligation. *J. Applied Logic*, 3(3-4):421–460, 2005.
4. J. Broersen, R. Wieringa, and J.-J. C. Meyer. A fixed-point characterization of a deontic logic of regular action. *Fundam. Inf.*, 48(2-3):107–128, 2001.
5. R. Bruni, H. C. Melgratti, and U. Montanari. Theoretical foundations for compensations in flow composition languages. In *POPL'05*, pages 209–220. ACM, 2005.
6. J. Carmo and A. J. Jones. Deontic logic and contrary-to-duties. volume 8, pages 265–343. Kluwer Academic Publishers, 2002.
7. P. F. Castro and T. S. E. Maibaum. A complete and compact propositional deontic logic. In *ICTAC'07*, volume 4711 of *LNCS*, pages 109–123. Springer, 2007.
8. R. M. Chisholm. Contrary-to-duty imperatives and deontic logic. *Analysis*, (XXIV):33–36, 1963.
9. K. Claessen. Safety property verification of cyclic synchronous circuits. In *SLAP'03*, volume 88 of *ENTCS*. Elsevier, 2003.
10. J. Coenen. Top-down development of layered fault tolerant systems and its problems- a denotic perspective. *Ann. Math. Artif. Intell.*, 9(1-2):133–150, 1993.

11. N. Dinesh, A. Joshi, I. Lee, and O. Sokolsky. A logic for regulatory conformance checking. In *Proceedings of the 14th Monterey Workshop*, 2007.
12. N. Dinesh, A. Joshi, I. Lee, and O. Sokolsky. Reasoning about conditions and exceptions to laws in regulatory conformance checking. In *DEON'08*, 2008. To appear.
13. S. Fenech, G. J. Pace, and G. Schneider. Conflict analysis of deontic contracts. In *WICT'08*, November 2008. To appear.
14. C. Hoare, M. Butler, and C. Ferreira. A trace semantics for long running processes. In *Communicating Sequential Processes: The First 25 Years*, volume 3525 of *LNCS*, pages 133–150. Springer, 2004.
15. S. Khosla and T. S. E. Maibaum. The prescription and description of state based systems. In *Temporal Logic in Specification*, volume 398 of *LNCS*, pages 243–294. Springer, 1987.
16. J. Li, H. Zhu, G. Pu, and J. He. A formal model for compensable transactions. In *ICECCS'07*, pages 64–73. IEEE Computer Society, 2007.
17. S. Liang, P. Hudak, and M. Jones. Monad transformers and modular interpreters. In *POPL '95*, pages 333–343. ACM, 1995.
18. A. Lomuscio and M. Sergot. Deontic interpreted systems. *Studia Logica*, (75):63–92, 2003.
19. P. McNamara. Deontic logic. volume 7 of *Handbook of the History of Logic*, pages 197–289. North-Holland Publishing, 2006.
20. J.-J. Meyer. Free choice permissions and ross's paradox: Internal vs. external nondeterminism. In *Proc. 8th. Amsterdam Collloquium*, pages 367–380. University of Amsterdam, 1992.
21. J.-J. C. Meyer. A different approach to deontic logic: Deontic logic viewed as a variant of dynamic logic. *Notre Dame Journal of Formal Logic*, 29:109–136, 1988.
22. R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., 1982.
23. O. Owe, G. Schneider, and M. Steffen. Components, objects, and contracts. In *SAVCBS'07*, ACM Digital Library, pages 91–94, Dubrovnik, Croatia, Sep. 2007.
24. M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, 40(11):38–45, 2007.
25. H. Prakken and M. Sergot. Contrary-to-duty obligations. *Studia Logica*, 57(1):91–115, 1996.
26. C. Prisacariu and G. Schneider. A formal language for electronic contracts. In *FMOODS'07*, volume 4468 of *LNCS*, pages 174–189. Springer, 2007.
27. C. Prisacariu and G. Schneider. Towards a formal definition of electronic contracts. Technical Report 348, Dept. of Informatics, Univ. of Oslo, Jan. 2007.
28. G. B. T. Shiple and H. Touati. Constructive analysis of cyclic circuits. In *European Design and Test Conference*, 1996.
29. W. van der Hoek, B. van Linder, and J.-J. C. Meyer. On agents that have the ability to choose. *Studia Logica*, 66(1):79–119, 2000.
30. G. H. V. Wright. Deontic logic. *Mind*, 60:1–15, 1951.
31. G. H. V. Wright. Deontic logic: A personal view. *Ratio Juris*, 12(1):26–38, 1999.
32. A. Z. Wyner. Sequences, obligations, and the contrary-to-duty paradox. In *DEON'06*, volume 4048 of *LNCS*, pages 255–271. Springer, 2006.