UNIVERSITY OF OSLO
Department of Informatics

$\mathcal{CL}$: A Logic for Reasoning about Legal Contracts —Semantics

Research Report No. 371

Cristian Prisacariu
Gerardo Schneider

Isbn 82-7368-330-3
Issn 0806-3036

February 2008

$\mathcal{CL}$ – A Logic for Reasoning about Legal Contracts: – Semantics [*]

Cristian Prisacariu[†] Gerardo Schneider[‡]

February 2008[§]

Abstract

The work reported here is concerned with the definition of a logic (which we call $\mathcal{CL}$) for reasoning about legal contracts. The report presents the syntax of the logic and the associated semantics. There are two semantics presented: one is defined with respect to linear structures (i.e. traces of actions) and is intended for run-time monitoring of executions of contracts; the second semantics is given over branching structures (i.e. Kripke-like structures) and is intended for reasoning about contracts in a static manner (i.e. model-checking and theorem proving). In the first part of the report we present the theoretical results underlying the branching semantics. It presents an algebra of actions and restates some of previous results presented in another report, as well as new results useful for the definition of the branching semantics and for the proofs. The rest of the report is concerned with the definition of the two semantics. Moreover, several (non-standard) desired properties of the logic are proven.

[*]Partially supported by the Nordunet3 project "COSoDIS – Contract-Oriented Software Development for Internet Services" (`http://www.ifi.uio.no/cosodis/`).

[†]Dept. of Informatics – Univ. of Oslo, P.O. Box 1080 Blindern, N-0316 Oslo, Norway. E-mail: cristi@ifi.uio.no

[‡]Dept. of Informatics – Univ. of Oslo, P.O. Box 1080 Blindern, N-0316 Oslo, Norway. E-mail: gerardo@ifi.uio.no

[§]Revised in March 2009.

Contents

1 Introduction **3**
- 1.1 Motivating the main features of $\mathcal{CL}$ 4
- 1.2 Example 10

2 $\mathcal{CL}$ – A Formal Language for Contracts **10**
- 2.1 Syntax of $\mathcal{CL}$ 10
- 2.2 Deontic Actions 13
- 2.3 Dynamic Actions 19

3 Direct semantics of $\mathcal{CL}$ **21**
- 3.1 Branching semantics in terms of normative structures 22
  - 3.1.1 Properties of the branching semantics 26
- 3.2 Trace semantics in terms of respecting traces 27
- 3.3 Relating the trace and the branching semantics 32

4 Conclusion **32**
- 4.1 Related Work 32

1 Introduction

With the advent of Internet-based development within e-business and e-government there is an increasing need to define well-founded theories to guarantee the successful integration and interoperability of inter-organizational collaborations. It is now widely accepted that in such complex distributed systems a *contract* is needed in order to determine what are the responsibilities and rights of the involved participants. Such a contract should also contain clauses stating what are the penalties in case of contract violations. Ideally, one would like to guarantee that the contract is contradiction-free by being able to reason about it, and to ensure that the contract is fulfilled once enacted. In order to do so the contract should be written in a formal language amenable to formal analysis; e.g. detection of contradictions/inconsistencies in contracts or superfluous clauses, or checking some desired properties on a contract. *Legal contracts*, as found in the usual judicial or commercial arena, may serve as basis for defining such machine-oriented *electronic contracts* (or e-contracts for short). This contracting style found in e-business and virtual organisations (and inspired from legal contracts) can also be used in service oriented architectures, component based systems [OSS07], and agent societies [CP01, vdT03]. In these areas contracts are used to regulate the interaction and exchanges between the parties involved (being that services, components, or agents).

Much research has been invested into giving a formalisation of contractual clauses, and also into providing a machine readable language for specifying contracts. Among the many approaches the most promising are the ones based on variants of deontic logic. Such a formal language is desired for doing static (like model-checking) or dynamic (like run-time monitoring) analysis of (abstractions of) contracts. Moreover, the automation of the negotiation process becomes a feasible goal.

We present a formal specification language for contracts, called $\mathcal{CL}$, able to represent obligations, permissions and prohibitions, as well as what happens when obligations or prohibitions are not respected. A first version of the language $\mathcal{CL}$ has been presented in [PS07b], where explicit temporal operators (always, eventually, and until) were part of the syntax. An encoding into a version of the modal mu-calculus with concurrent actions was used to give semantics. The $\mathcal{CL}$ language presented in this paper is more expressive and has a cleaner syntax (with no syntactic restrictions). A variant of this syntax of the $\mathcal{CL}$ (without the propositional constants) was used in [KPS08a] for doing run-time monitoring of electronic contracts. In [KPS08a] it was used a restricted semantics based on traces of actions. This semantics was specially designed for monitoring the actions of the contracting parties at run-time with the purpose of detecting when a contract is violated. The presentation of [KPS08a] did not give much explanation nor examples about the intuitions behind the choices in the design of $\mathcal{CL}$. We do this in the present paper and discuss the full semantics of $\mathcal{CL}$ based on *normative structures*. We focus on the intended properties of the language. We do not present the notions formally.

The goal of $\mathcal{CL}$ is to faithfully capture several concepts used in the context of electronic contracts, to avoid deontic paradoxes, and to preserve many of the natural properties relevant in legal contracts. Since our main objective is to analyse contracts through formal verification techniques, we aim at a tractable language (i.e. decidable and with manageable complexities). In this way, we could use formal tools like model-checking and run-time monitoring.

The report is organized as follows. The rest of the introduction gives intuitions about the main features of $\mathcal{CL}$ accompanied by example from a contract between an Internet provider and a Client given in Figure **??**.

In Section **??** we present the algebra of concurrent actions (capturing the intuitions of actions found in contracts). The actions are interpreted as specially defined guarded rooted trees which are used in the semantics of $\mathcal{CL}$. We state the main result of this section which is the completeness of the algebra over rooted trees. The rest of the section is concerned with the extension of the algebra with boolean tests and action negation (which is defined using a canonical form of the actions).

In Section 3 we construct two direct semantics for the $\mathcal{CL}$ language. The first semantics is a branching semantics given in terms of particular Kripke structures (which we call *normative structures*). The branching semantics is intended to provide for reasoning about the contracts written in $\mathcal{CL}$; particularly model checking of contracts, or negotiation of contracts should be performed using this semantics. Proof systems for $\mathcal{CL}$ should be based on the branching semantics. We do not provide any of those in this report; we refer the reader to future work. The second semantics is defined in terms of linear models represented by traces of actions. The linear semantics captures the notion of *a trace fulfills a contract.* The linear semantics is used for run-time monitoring of the enforcement of contracts written in $\mathcal{CL}$.

1.1 Motivating the main features of $\mathcal{CL}$

In this section we motivate the particular choices we made in the design of the $\mathcal{CL}$ contract specification language. We compare to related works and give informal intuitions and examples.

The purpose of $\mathcal{CL}$ is to specify and reason about contracts, therefore $\mathcal{CL}$ integrates the normative notions of *obligation*, *permission*, and *prohibition.* These have been extensively investigated in deontic logic [Wri51] which is a modal logic where only the **K** and **D** axioms hold and not the **T** axiom. The deontic notions that we introduce in $\mathcal{CL}$ are different than the ones in standard deontic logic (SDL) in several respects.

(1) The deontic modalities are applied only over actions instead of over propositions (or state of affairs) as in SDL. This is known as the ought-to-do approach to deontic logic as opposed to the more classic ought-to-be approach of SDL. The ought-to-do approach has been advocated by vor Wright [VW68] which argued that deontic logic would benefit from a "foundation of actions", since many of the philosophical paradoxes of SDL would not occur. Important contributions to this approach were done by Segerberg for introducing the actions inside the deontic modalities [Seg82, Seg92] and by the seminal work of Meyer on dynamic deontic logic (DDL) [Mey88] (see also [BWM01, Wyn06]).

Compared to [Mey88, BWM01, VdM90, CM07], which also consider deontic modalities (i.e. O, P, and F) applied over actions, the investigation presented in this paper at the level of the actions is different in several ways:

(2) The action combinators are the standard $+$ and $\cdot$ (for *choice* and *sequence*) but exclude the Kleene star *. None of the few papers that consider repetition (e.g. using *) as an action combinator under deontic modal-

ities [VdM90, BWM01] gives a precise motivation for having such recurring actions inside obligations, permissions, or prohibitions. In fact its use inside the deontic modalities seems counter intuitive: take the expression $O(a^*)$ - "One is obliged to not pay, or pay once, or pay twice in a row, or..." - which puts no actual obligations; or take $P(a^*)$ - "One has the right to do any sequence of action a." - which is a very shallow permission and is captured by the widespread *Closure Principle* in jurisprudence where *what is not forbidden is permitted* [Seg82]. Moreover, as pointed out in [BWM01] expressions like $F(a^*)$ and $P(a^*)$ can be simulated with the propositional dynamic logic (PDL) modalities as $\langle a^*\rangle F(a)$ respectively $[a^*]P(a)$. In our opinion the * combinator under deontic modalities can be captured by using temporal or dynamic logic modalities along with deontic modalities over actions without the Kleene star * (as we will see later).

(3) We add a concurrency operator $\times$ to model that two actions are *done at the same time*. The model of concurrency that we addopt is the synchrony model of Milner's SCCS [Mil83]. Synchrony is easy to integrate with the other regular operations on actions (the choice and the sequence). Moreover, synchrony is a natural choice when reasoning about the notion "at the same time" for human-like actions that we have in legal contracts (opposed to the instructions in a programming language).

The notion of synchrony has different meanings in different areas of computer science. Here we take the distinction between *synchrony* and *asynchrony* as presented in the SCCS calculus and later implemented in e.g. the Esterel synchronous programming language [Ber00]. We understand *asynchrony* as when two concurrent systems proceed at indeterminate relative speeds (i.e. their actions may have different non-correlated durations); whereas in the *synchrony* model each of the two concurrent systems instantaneously perform a single action at each time instant. This is an abstract view of the actions found in contracts which is good for reasoning about properties of the contract. If one needs actions which have durations (e.g. "work 3 hours") or which are parameterised by amounts (e.g. "deposit 100$") then $\mathcal{CL}$ has to be extended accordingly.

The *synchrony model* of concurrency takes the assumption that time is discrete and that basic actions are instantaneous (i.e. take zero time and represent the time step). Moreover, at each time step all possible actions are performed, i.e. the system is considered *eager* and *active*. For this reason if at a time point there is enabled an obligation to do an action then this action must be immediately executed so that the obligation is not violated. Synchrony assumes a global clock which provides the time for all the actors in the system. Note that for practical implementation purposes this is a rather strong assumption which offends the popular view from process algebras [Mil95, Hoa85]. On the other hand the mathematical framework of the synchrony model is much cleaner and more general than the asynchronous interleaving model (SCCS has the (asynchronous) CCS as a subcalculus [Mil83]). The synchronous composition operator $\times$ is different from the classical $||$ of CCS.

The synchrony model is better suited for *reasoning* about concurrent actions than for implementing true concurrency. Because of the assumption of an eager behaviour for the actions the scope of the obligations (and of the other deontic modalities too) is immediate, making them transient obligations which are enforced only in the current world. One can get persistent obligations by using temporal operators, like the box operator *always*. The eagerness assumption facilitates both reasoning about existence of the deontic modalities and about violations of the obligations or prohibitions.

(4) We define a *conflict relation* $\#_\mathcal{C}$ over actions which represents the fact that two actions cannot be done at the same time. This is necessary for detecting (and for ruling out) a first kind of *conflicts* in contracts: "Obligatory to go west and obligatory to go east" should result in a conflict. The second kind of conflicts that the $\mathcal{CL}$ rules out are: "Obligatory to go west and forbidden to go west" which is a standard requirement on a deontic logic.

(5) $\mathcal{CL}$ defines an *action complement* operation which encodes the violation of an obligation. Obligations (and prohibitions) can be violated by *not* doing the obligatory action (or *doing* the forbidden action). The action complement that we have is different from the various notions of action negation found in the literature on PDL or DDL-like logics [Mey88,HKT00,LW04,Bro03]. In [Mey88], the same as in [HKT00] action negation is with respect to the universal relation which for PDL gives undecidability. Decidability of PDL with negation of only atomic actions has been achieved in [LW04]. A so called "relativized action complement" is defined in [Bro03] which is the complement of an action (not w.r.t. the universal relation but) w.r.t. a set formed of atomic actions closed under the application of the action operators. This kind of negation still gives undecidability when several action operators are involved.
In $\mathcal{CL}$ the *action complement* is a derived operator denoted by $\overline{\alpha}$ and is defined as a function which takes the compound action α and returns another compound action (i.e. action complement is not a principal combinator for the actions like $+$, $\cdot$, or $\times$). Intuitively action complement $\overline{\alpha}$ is the action given by *all* the immediate actions that *take us outside* the tree of α [BWM01].
The formalisation of the actions has been thoroughly investigated in [Pri08b] where standard models have been defined for the actions, and completeness and decidability results have been established. The semantics of the $\mathcal{CL}$ language is based on this interpretation of the synchronous actions as special trees.

In what follows we consider other particularities of the $\mathcal{CL}$ language which are not necessarily related to the actions. As we hinted before, $\mathcal{CL}$ combines notions from both deontic logic and propositional dynamic logic. $\mathcal{CL}$ is an *action-based* language therefore, the logic of choice when we want to talk about actions (what happens *after* an action is performed) is PDL. Because of its particular application to legal contracts $\mathcal{CL}$ incorporates several notions found in distinct variations of PDL. We take the *dynamic modality* $[\beta]\mathcal{C}$ which is read as: "after the action β is performed then the formula $\mathcal{C}$ (in our case a contract clause) should be the case (in the next, changed, world)".

(6) One difference from the standard PDL is that we consider *deterministic* actions. This is natural and desired in legal contracts as opposed to the programming languages community where nondeterminism is an important notion. In contracts the outcome of an action like "deposit 100$ in the bank account" is uniquely determined. The deterministic PDL has been investigated in [BAHP81]. On the other hand deterministic PDL is undecidable if action negation (or intersection of actions) is added [HKT00].

(7) Another important feature of $\mathcal{CL}$ is the introduction of the synchrony operation $\times$ on the actions inside the dynamic modality. Therefore, $\mathcal{CL}$ can reason about synchronous actions and from this point of view it is included

in the class of extensions of PDL which can reason about concurrent actions: PDL$^\cap$ with intersection of actions [Har83] which is undecidable for deterministic structures; or concurrent PDL [Pel85,Pel87] which adopts ideas from alternating automata [CKS81].

Contrasting with the discouraging results of undecidability of the various logics from above, $\mathcal{CL}$ (with action complement and synchronous composition over deterministic actions inside the dynamic modality) is decidable. This makes $\mathcal{CL}$ more attractive for automation of reasoning about contracts.

(8) In $\mathcal{CL}$ *conditional obligations* (or prohibitions) can be of two kinds.

a. The first kind is given with the propositional implication: $\mathcal{C} \rightarrow O_{\mathcal{C}}(\alpha)$ which is read as "if $\mathcal{C}$ is the case then it is obligatory that action α". As an example, let us consider the clause **??** of the contract in Fig. **??**: "If Internet traffic is high then the Client is obliged to pay".

b. The second kind is given with the dynamic box modality: $[\beta]O_{\mathcal{C}}(\alpha)$ which is read as "if action β was performed then it is obligatory that action α". As an example from the contract of Fig. **??** take clause **??** (rewarded for simplicity): "after receiving necessary data ... the Provider is obliged to offer ... password".

(9) Regarding the deontic modalities, $\mathcal{CL}$ includes directly in the definition of the obligation and prohibition the *reparations* in case of violations. The deontic modalities are $O_{\mathcal{C}}$ and $F_{\mathcal{C}}$ where $\mathcal{C}$ is a contract clause representing the reparation. This models the notions of contrary-to-duty obligations (CTDs) and contrary-to-prohibitions (CTPs) as found in deontic logic applied over actions like DDL [Mey88, Wyn06]. These notions are in contrast with the classical notion of CTD as found in the SDL literature [PS97, CJ02]. In SDL, what we call reparations are secondary obligations which hold in the same world as the primary obligation. In our setting where the action changes the context (the world) one can see a violation of an obligation (or prohibition) only after the action is performed and thus the reparations are enforced in the next world (in the changed context).

The approach of $\mathcal{CL}$ to contrary-to-duty rules out many of the problems faced by SDL (like the gentle murderer paradox). On the other hand it does not capture the wording of the SDL examples.

(10) The deontic modalities are *not interdefinable* in $\mathcal{CL}$. Only some of the implications hold and are discussed next.

(11) The *semantics* of $\mathcal{CL}$ is given in terms of *normative structures* and it is specially defined to capture several natural properties which are found in legal contracts.

a. One of the interesting properties relates the existence of two obligations with the synchrony operation: $O_{\mathcal{C}}(\alpha) \wedge O_{\mathcal{C}}(\beta) \rightarrow O_{\mathcal{C}}(\alpha \times \beta)$. The formula should be read as: If "there exists an obligation to do action α and there is also an obligation to do action β" (in the same current world) then we should be able to infer that "there is an obligation to do both actions α and β at the same time". Otherwise one of the two obligations would be violated.[1]

Related to each property we give examples taken (or adapted) from the contract of Fig. **??**. For the property (a) above consider e.g.: "Client is obliged to delay payment" and also "Client is obliged to notify by e-mail" then we can conclude that "Client is obliged to delay and notify at the same time".

Other properties which come from SDL are:

b. Obligation of an action implies that the action is permitted:
$$O_{\mathcal{C}}(\alpha) \rightarrow P(\alpha)$$
E.g.: "Client is obliged to pay" then "Client has the right to pay".

c. Permission of performing an action implies that the action is not forbidden:
$$P(\alpha) \rightarrow \neg F_{\mathcal{C}}(\alpha)$$
E.g.: "Provider has the right to alter personal data" then "Provider is not forbidden to alter personal data".

d. If two action expressions represent the same action then the obligation of one action should imply the obligation of the other action:
$$\text{if } \alpha = \beta \text{ then } O_{\mathcal{C}}(\alpha) \leftrightarrow O_{\mathcal{C}}(\beta)$$

There are several properties particularly related to the ought-to-do approach (deontic modalities applied over actions); some can be found in DDL-like logics too:

d. Obligation of the sequence of two actions α followed by β implies the obligation of the first action α and after the first action is performed then the obligation of the second action β is enforced:
$$O_{\mathcal{C}}(\alpha \cdot \beta) \leftrightarrow O_{\mathcal{C}}(\alpha) \wedge [\alpha]O_{\mathcal{C}}(\beta)$$
E.g.: The clause "Client is obliged to lower the Internet traffic and then to pay", implies that "Client is obliged to lower the Internet traffic (now)" and "After Client lowers the Internet traffic then Client is obliged to pay".

e. Prohibition of an action α implies that any action that is bigger (i.e. includes) action α is prohibited:[2]
$$F_{\mathcal{C}}(\alpha) \rightarrow F_{\mathcal{C}}(\alpha \times \beta)$$
E.g.: "Client is forbidden to supply false information" then we also know that "Client is forbidden to supply false information and at the same time supply correct information".

We comment more on this property. One may think of an example like (in an idealised society) "One is forbidden to smoke" but still "One is permitted to smoke and (at the same time) sit outside in the open air". This example seems to contradict the above property. The confusion comes

[1] The properties that we now talk about intuitively are validities in the logic which means that they are *always* true (in any normative structure at any world).

[2] Take concurrent actions, which specify that two or more actions are done at the same time; e.g. "drink and drive at the same time". One concurrent action is considered *bigger* than another concurrent action if and only if all the actions in the smaller concurrent action are specified in the bigger action too; e.g. "drink and drive and talk to mobile at the same time" is bigger than "drink and drive at the same time".

from the wording of the above example. A more correct wording would be: “One is forbidden to smoke and (at the same time) sit in a public place” where this action is no longer smaller than the action “smoke and (at the same time) sit outside in the open air” and thus the prohibition and the permission go along together.

f. The prohibition of a sequence of two actions α followed by β is equivalent to the prohibition of the first action α or otherwise after the first action is performed then there is the prohibition of the second action β:

$$F(\alpha \cdot \beta) \leftrightarrow F(\alpha) \vee \langle\alpha\rangle F(\beta)$$

The intuition for this property is that the prohibition of a sequence of actions hods if and only if there exists at least one of the actions at some point in the sequence that is forbidden. A natural consequence of this property is that if “One is forbidden to smoke” then also “One is forbidden to smoke and after that go to some open air place”. On the other hand if “One is forbidden to drink and after that drive” this does not imply that “One is forbidden to drink” nor that “One is forbidden to drive”.

g. The prohibition of a choice of two actions α or β is the same as having both prohibition of α and prohibition of β in the same world:

$$F_{\mathcal{C}}(\alpha + \beta) \leftrightarrow F_{\mathcal{C}}(\alpha) \wedge F_{\mathcal{C}}(\beta)$$

E.g.: “Client is forbidden to pay in dollars or to pay in euro” this means that “Client is forbidden to pay in dollars” and that “Client is forbidden to pay in euro”.

h. Two similar properties can be proven for permissions. Permission of a sequence is the same as permission of all the actions in the sequence. Permission of a choice of actions is the same as permission of all the actions in the choice.

In the design decisions for $\mathcal{CL}$ we give special attention to what we call *unwanted implications*. This kind of “properties” are rather scarce and neglected in the literature.

i. Related to (e), if there is the prohibition of doing two actions α and β at the same time this does not imply that any of the two actions is prohibited (i.e. the converse implication of (e) does not always hold).

$$F_{\mathcal{C}}(\alpha \times \beta) \not\rightarrow F_{\mathcal{C}}(\alpha)^3$$

E.g.: “One is prohibited to drink and drive at the same time” does not imply that “One is forbidden to drink” and neither that “One is forbidden to drive”.

j. Similarly with permissions: If there is the permission to do two actions at the same time this does not imply that any of the two actions is permitted.

$$P(\alpha \times \beta) \not\rightarrow P(\alpha)$$

E.g.: “One is permitted to smoke and sit outside in open air” does not imply that “One is permitted to smoke” because if one sits inside a restaurant then one is forbidden to smoke.

[3] Where the sign $\not\rightarrow$ should be understood as *the implication is not valid*, i.e. $\not\models$ where $\models$ is the validity symbol. We consider that $\not\rightarrow$ is more intuitive and we do not need to define the validity and non-validity signs here.

k. Obligation of an action α does not imply obligation of any concurrent action that contains α. Similarly, obligation of a concurrent action does not imply obligation of any of its composing actions.

$$O_{\mathcal{C}}(\alpha) \not\to O_{\mathcal{C}}(\alpha \times \beta);$$
$$O_{\mathcal{C}}(\alpha \times \beta) \not\to O_{\mathcal{C}}(\alpha).$$

E.g.: "Obligation to drive" should not imply "Obligation to drive and drink at the same time". For the second unwanted implication consider "Obligation to smoke and sit outside" which should not imply "Obligation to smoke".

l. Obligation of an action α does not imply the obligation of the choice between that action and any other. Similarly, obligation of a choice of actions does not imply obligation of any of the actions in the choice.

$$O_{\mathcal{C}}(\alpha) \not\to O_{\mathcal{C}}(\alpha + \beta);$$
$$O_{\mathcal{C}}(\alpha + \beta) \not\to O_{\mathcal{C}}(\alpha).$$

E.g.: "Client is obliged to pay or to delay payment" should not imply that "Client is obliged to delay payment". For the second unwanted implicaiton "Obliged to mail the letter" should not imply "Obliged to mail the letter or burn the letter".

m. As consequence, obligation of a choice of two actions does not imply obligation of the two actions done at the same time. Analogously, obligation to do two actions at the same time does not imply obligation of the choice of the two actions.

$$O_{\mathcal{C}}(\alpha + \beta) \not\to O_{\mathcal{C}}(\alpha \times \beta);$$
$$O_{\mathcal{C}}(\alpha \times \beta) \not\to O_{\mathcal{C}}(\alpha + \beta).$$

1.2 Example

We use throughout the report the following small example of a contract clause in order to exemplify some of the main concepts we introduce.

Example 1.: "If the *Client* exceeds the bandwidth limit then (s)he must pay [*price*] immediately, or (s)he must delay the payment and notify the *Provider* by sending an e-mail. If in breach of the above (s)he must pay double."

2 $\mathcal{CL}$ – A Formal Language for Contracts

2.1 Syntax of $\mathcal{CL}$

$\mathcal{CL}$ is an *action-based* language. The syntax of $\mathcal{CL}$ is defined by the grammar in Table 1. In what follows we provide intuitions of the $\mathcal{CL}$ syntax and define our notation and terminology.

We call an expression $\mathcal{C}$ a (general) *contract clause.* An expression may be a propositional constant (or assertion) φ drawn from a finite set Φ_B. We call $O_{\mathcal{C}}(\alpha)$, $P(\alpha)$, and $F_{\mathcal{C}}(\alpha)$ the *deontic modalities*, representing the obligation, permission, or prohibition of performing a given *action* α. Intuitively $O_{\mathcal{C}}(\alpha)$ states the obligation to perform α, and the *reparation* $\mathcal{C}$ in case the obligation is *violated*, i.e. whenever α is not performed. The reparation may be any contract clause. The modality $O_{\mathcal{C}}(\alpha)$ (resp. $F_{\mathcal{C}}(\alpha)$) represents what is called CTD (resp. CTP) in dynamic deontic logic. Obligations without reparations are

$$
\begin{array}{rcl}
\mathcal{C} & := & \varphi \mid O_{\mathcal{C}}(\alpha) \mid P(\alpha) \mid F_{\mathcal{C}}(\alpha) \mid \mathcal{C} \rightarrow \mathcal{C} \mid [\beta]\mathcal{C} \mid \bot \\
\alpha & := & a \mid \mathbf{0} \mid \mathbf{1} \mid \alpha \& \alpha \mid \alpha \cdot \alpha \mid \alpha + \alpha \\
\beta & := & a \mid \mathbf{0} \mid \mathbf{1} \mid \beta \& \beta \mid \beta \cdot \beta \mid \beta + \beta \mid \beta^* \mid \varphi? \\
\varphi? & := & \varphi \mid \mathbf{0} \mid \mathbf{1} \mid \varphi? \vee \varphi? \mid \varphi? \wedge \varphi? \mid \neg\varphi?
\end{array}
$$

Table 1: Syntax of the contract language $\mathcal{CL}$.

written as $O_\bot(\alpha)$ where $\bot$ (and conversely $\top$) is the Boolean `false` (respectively `true`). We usually write $O(\alpha)$ instead of $O_\bot(\alpha)$. Obligations with no reparation are sometimes in the literature called *categorical* because they must not be violated (i.e. there is no reparation for their violation, thus a violation would give violation of the whole contract). The prohibition modality $F_{\mathcal{C}}(\alpha)$ states the actual forbearing of the action α together with the reparation $\mathcal{C}$ in case the prohibition is violated. Note that it is possible to express nested CTDs and CTPs. Permissions have no reparations associated because they cannot be violated; permissions can only be exercised.

We use the classical Boolean implication operator $\rightarrow$, and the other operators $\wedge, \vee, \neg, \leftrightarrow, \top, \oplus$ (exclusive or) are expressed in terms of $\rightarrow$ and $\bot$ as in propositional logic.

The dynamic logic modality $[\cdot]\mathcal{C}$ is parameterised by *actions* β. The expression $[\beta]\mathcal{C}$ is read as: "after the action β is performed $\mathcal{C}$ must hold".

Notice that it is possible to express nested CTDs and CTPs. Nested CTDs (or CTPs) produce *finite* chains of CTDs (respectively CTPs). Because we lack explicit recursion (e.g. like μ-calculus has [Koz83]) in our syntax the chains cannot be *infinite*. For infinite chains (where one obligation can be the reparation of itself, or there can be circular reparations...) we can use the unbounded recursion μ (or the infinite recursion with ν) of μ-calculus. For example we write $\mu X.O_{O_X(\beta)}(\alpha)$ to express two mutually reparation obligations (i.e. two obligations where the first is the reparation of the second and the second is the reparation of the first). This example gives an infinite chain of CTDs.

Propositional dynamic logic (PDL) makes an interplay between the actions and the formulas, and interdefines the two; i.e. it has formulas as actions (the tests) and it has actions defining the formulas (the box modality). Note the two differences between the actions β which appear inside the PDL modality $[\cdot]$ and the actions α which are allowed inside the deontic modalities. We argued against not having the Kleene * for the α actions. We argue now about not having the test actions inside the deontic modalities.

The intuition of the *test action* is that the action $\varphi?$ can be performed only if the formula φ holds in the current world. The same, the sequence action $\varphi? \cdot \alpha$ can be viewed as a *guarded* action because the α can be performed only if the test $\varphi?$ succeeds. If we allow the *test actions* $\varphi?$ inside the deontic modalities it would break the ought-to-do approach because they introduce the formulas inside the action formalism. Therefore, we would have obligations applied over formulas; e.g. $O_{\mathcal{C}}(\varphi?)$. This would constitute in a combination of ought-to-do and ought-to-be (for this direction check [dMW96]). Moreover, adding the tests inside the deontic modalities does not integrate with our way of giving semantics; we do not know how to mark obligatory (or prohibited) tests, as we do with the actions. Therefore, the reparation $\mathcal{C}$ is enforced in the same world as the O and the F of the example. There is no state change. Therefore we could reason only with the propositional logic part of $\mathcal{CL}$ as we see below.

Suppose for the sake of example that we take this interplay between the tests and the deontic modalities. Take the example above $O_{\mathcal{C}}(\varphi?)$ which is read as "It is obligatory (in the current world) that the test $\varphi?$ holds (in the current world). Otherwise (if the test does not hold) the reparation $\mathcal{C}$ should be enforced afterwards.". Note that the actions change the world. On the other hand the tests do not change the world: if a tests succeeds then we remain in the same world, if the test fails then the whole action fails (is equivalent to the violating action $\mathbf{0}$ or to the unsatisfiable special formula $\perp$). We can achieve the same by only using the $\mathcal{CL}$ language as we have it. The example above is specified in $\mathcal{CL}$ as $\varphi \vee (\neg\varphi \wedge \mathcal{C})$ which is read as above (we reword it here to match the formula better): "(In the current world) φ must hold or otherwise φ does not hold and the formula $\mathcal{C}$ holds. Note the difference of our formula and the formula $\varphi \vee \mathcal{C}$ which does not capture what we want.

Propositional dynamic logic can encode the temporal logic (TL) [Pnu77] operators $\mathcal{U}$ (*until*), $\bigcirc$ (*next*), $\Box$ (*always*), and $\Diamond$ (*eventually*). The special action $\mathbf{any} = +_{\alpha_\times \in \mathcal{A}_B^\times} \alpha_\times$ is a shortcat which stands for the finite choice between *all* concurrent actions of $\mathcal{A}_B^\times$. Thus $\mathcal{C}_1 \, \mathcal{U} \, \mathcal{C}_2$ (in temporal logic states that $\mathcal{C}_1$ holds until $\mathcal{C}_2$ holds) stands for $\langle(\mathcal{C}_1? \cdot \mathbf{any})^*\rangle\mathcal{C}_2$ (which is read in PDL, equivaletly as in TL, as there exists a state where $\mathcal{C}_2$ holds and is reached by going through states where $\mathcal{C}_1$ holds). Note that $\mathcal{C}_2$ might hold in the current state. $\bigcirc\mathcal{C}$ stands for $[\mathbf{any}]\mathcal{C}$ and intuitively states that $\mathcal{C}$ holds in the next moment, usually after something happens. $\Box\mathcal{C}$ stands for $[\mathbf{any}^*]\mathcal{C}$ and expresses that $\mathcal{C}$ holds in every moment. $\Diamond\mathcal{C}$ stands for $\langle\mathbf{any}^*\rangle\mathcal{C}$ expressing that $\mathcal{C}$ holds sometimes in a future moment. See [HKT00, sec.17.2] for more on embedding linear temporal logic operators into propositional dynamic logic.

Remark: A natural contract clause would be that "obligation to do the sequence of actions a and then b holds until some constraint φ is satisfied" which one would write in a temporal logic style like $O(a \cdot b)\mathcal{U}\varphi$. Because the $\mathcal{U}$ does not behave nicely when coupled with an operator over sequences of actions we write the same clause nicely in our $\mathcal{CL}$ syntax like $\langle((\neg\varphi \wedge O(a \cdot b))? \cdot a \cdot b)^*\rangle\varphi$.

The purpose of the $\mathcal{CL}$ language is to give a unified framework for writing both the original contract clauses and also contract properties that need to be checked on the contract model. The syntax that we give in Table 1 is rather general and unrestricted. This is good for writing properties about contracts but for the specification of the contracts clauses this syntax may require syntactic restrictions. Note that an asserion ϕ alone may be a contract clause. This is normal and desirable when one wants to write properties about a contract like "The budget is more than...". On the other hand it is not normal (and one does not find in the legal contracts) to have an assertion as a stand alone contract clause. In a contract, assertions are used in a restricted fashon. A first restriction could be that in the contract clauses assertions are allowed only after the dynamic box (i.e. after the execution of an action so that it asserts the outcome of that action). The semantics of the operators remains the same, only the allowed $\mathcal{CL}$ formulas are different.

In this paper we are interested in the general properties of the legal notions and therefore we investigate the general $\mathcal{CL}$. We do not investigate syntactic restrictions which might be desired for a practical implementation of $\mathcal{CL}$.

(1) $\alpha + (\beta{+}\gamma) {=} (\alpha{+}\beta){+}\gamma$	(10) $\alpha{\times}(\beta{\times}\gamma) = (\alpha{\times}\beta){\times}\gamma$
(2) $\alpha + \beta = \beta + \alpha$	(11) $\alpha{\times}\beta = \beta{\times}\alpha$
(3) $\alpha + \mathbf{0} = \mathbf{0} + \alpha = \alpha$	(12) $\alpha{\times}\mathbf{1} = \mathbf{1}{\times}\alpha = \alpha$
(4) $\alpha + \alpha = \alpha$	(13) $\alpha{\times}\mathbf{0} = \mathbf{0}{\times}\alpha = \mathbf{0}$
(5) $\alpha \cdot (\beta \cdot \gamma) = (\alpha \cdot \beta) \cdot \gamma$	(14) $a{\times}a = a \quad \forall a \in \mathcal{A}_B$
(6) $\alpha \cdot \mathbf{1} = \mathbf{1} \cdot \alpha = \alpha$	(15) $\alpha{\times}(\beta + \gamma) = \alpha{\times}\beta + \alpha{\times}\gamma$
(7) $\alpha \cdot \mathbf{0} = \mathbf{0} \cdot \alpha = \mathbf{0}$	(16) $(\alpha + \beta){\times}\gamma = \alpha{\times}\gamma + \beta{\times}\gamma$
(8) $\alpha{\cdot}\,(\beta{+}\gamma){=}\alpha{\cdot}\,\beta + \alpha{\cdot}\,\gamma$	(17) $(\alpha_\times \cdot \alpha){\times}(\beta_\times \cdot \beta) =$
(9) $(\alpha{+}\beta){\cdot}\,\gamma = \alpha{\cdot}\,\gamma + \beta \cdot \gamma$	$(\alpha_\times{\times}\beta_\times){\cdot}(\alpha{\times}\beta)\ \forall \alpha_\times, \beta_\times \in \mathcal{A}_B^\times$

Table 2: Axioms of action equality.

2.2 Deontic Actions

Definition 2.1. *Consider a finite set of* basic *(or atomic) actions $\mathcal{A}_B$ (denoted by $a, b, c, \ldots$). The special actions $\mathbf{0}, \mathbf{1} \notin \mathcal{A}_B^\times$ are called respectively the* violating *and the* skip *actions. The action combinators are: "$+$" for* choice *of two actions, "$\cdot$" for* sequence *of two actions (or concatenation), "$\times$" for* concurrent composition *(synchronously) of two actions. We generally call* compound actions *(or just actions) terms of $\mathcal{A}$ (denoted by indexed α) obtained from basic actions, $\mathbf{0}$, and $\mathbf{1}$ using the action combinators. We call* concurrent actions*, denoted $\alpha_\times$, the subset of actions $\mathcal{A}_B^\times \subset \mathcal{A}$ generated from $\mathcal{A}_B$ using only $\times$ constructor. To avoid unnecessary parentheses we use the following precedence over the combinators: $+ < \cdot < \times$. Table 2 axiomatizes the equality between actions.*

Actions as presented here are related to the more general algebraic structure called *synchronous Kleene algebra* in [Pri08b]. Note that $\mathbf{0}, \mathbf{1} \notin \mathcal{A}_B^\times$. Also note that $\mathcal{A}_B^\times$ is finite because $\mathcal{A}_B$ is finite and $\times$ is idempotent over basic actions; see axiom (14) Note the inclusion of sorts $\mathcal{A}_B \subseteq \mathcal{A}_B^\times \subset \mathcal{A}$. The axioms (1)-(4) define a commutative and idempotent monoid for $+$. Axioms (5)-(7) define a monoid with an annihilator element for the $\cdot$ sequence combinator. Axioms (8) and (9) give the distributivity of $\cdot$ over $+$. These give the algebraic structure of an idempotent semiring $(\mathcal{A}, +, \cdot, \mathbf{0}, \mathbf{1})$.

Axioms (10)-(13) make $(\mathcal{A}, \times, \mathbf{1}, \mathbf{0})$ a commutative monoid with an annihilator element. Axiom (14) defines $\times$ to be idempotent over the basic actions $a \in \mathcal{A}_B$. Therefore, we take the liberty of using $\subseteq$ to compare elements of $\mathcal{A}_B^\times$, as $\alpha_\times$ can be seen as the set of basic actions that compose it (e.g. $a{\times}b \subseteq a{\times}b{\times}c$ or $a \not\subseteq b{\times}c$). Axioms (15) and (16) define the distributivity of $\times$ over $+$. From axioms (10)-(16) together with (1)-(4) we conclude that $(\mathcal{A}, +, \times, \mathbf{0}, \mathbf{1})$ is a commutative and idempotent semiring (NB: idempotence comes from (4), whereas (14) is an extra property of the semiring).

At this point we give an informal intuition for the elements (actions) of $\mathcal{A}$: we consider that the actions are performed by somebody (being that a person, a program, or an e-agent). We talk about "doing" and one should not think of *processes "executing" actions* and operational semantics like in SCCS; we do not discuss operational semantics nor bisimulation equivalences in this paper.

Definition 2.2 (demanding relation)**.** *We call $<_\times$ the demanding relation and define it as:*

$$\alpha <_\times \beta \triangleq \alpha{\times}\beta = \beta \tag{1}$$

We say that β is more demanding than α iff $\alpha <_\times \beta$. We denote by $\leq_\times$ the relation $<_\times \cup =$; i.e. $\alpha \leq_\times \beta$ iff either $\alpha <_\times \beta$ or $\alpha = \beta$.

Note that the least demanding action is $\mathbf{1}$ (skiping means not doing any action). On the other hand, if we do not consider $\mathbf{1}$ then we have the basic actions of $\mathcal{A}_B$ as the least demanding actions; the basic actions are not related to each other by $<_\times$. It is routine to prove that the relation $<_\times|_{\mathcal{A}_B^\times}$ (i.e. $<_\times$ restricted to concurrent actions) is a partial order. Consider the following examples: $\mathbf{1} <_\times a$, $a <_\times a \times b$, $a <_\times a$, $a + b \not<_\times a + b$, $a \not<_\times b$, and $a \not<_\times b \times c$.

Remark: The *intersection* operator $\cap$ over actions form PDL$^\cap$ (i.e. PDL extended with intersection of actions; see [HKT00, sec.10.4]) is interpreted as intersection of relations (corresponding to the actions). Therefore $\cap$ respects all our axioms (10)-(16) except (12); i.e. it is associative, commutative, has the empty relation as anihilator element, is idempotent, and is distributive over the union of relations $\cup$ (as in PDL$^\cap$ the choice of actions $+$ is interpreted as union of relations). Our axiom (12) shows a difference between the interpretation that we want for $\times$ and the interpretation that PDL$^\cap$ gives to $\cap$. The identity element of $\cup$ is the universal relation and not the identity relation (which is the interpretation of $\mathbf{1}$).

Note that axiom (14) is given only over basic actions. If we were to give this over general actions α as is the case in PDL$^\cap$, in our algebra the concurrency operator $\times$ may have unwanted behavior because of the combination of idempotency and distributivety. That means that reflexivity is not a desired property of the general actions, but only of the concurrent actions. Therefore, ireflexivity is not a property of the general actions either. For example $(a + b) \times (a + b) = a + b + a \times b$ but on the other hand $(a + b + a \times b) \times (a + b + a \times b) = a + b + a \times b$ due to the idempotence of the $\times$ over $\mathcal{A}_B$. Moreover, we can prove that for any action α there exists a fix point for the application of the $\times$ to the action itself. In other words, define $\beta_0 = \alpha$ and $\beta_i = \beta_{i-1} \times \alpha$, then $\exists n \in \mathbb{N}$ and $\exists j < n$ s.t. $\beta_j = \beta_n$. The proof uses the canonical representation of the action α (from Definition 2.4).

In the case of intersection $\cap$ this problem does not occure because of the absorbtion property of the $\cup$ which states that:

$$A \cup (A \cap B) = A$$

which is the same as saying that if $C \subseteq D$ then $C \cup D = D$.

In the case of actions found in contracts we decide not to have absorbtion as it is not natural. In our case absorbtion is written as: if $\alpha <_\times \beta$ then $\alpha \times \beta = \alpha$. This means that when choosing between $a + (a \times b)$ one would always have to choose the *least demanding* action which is a in this case. We would call the least demanding action the *most preferable*. In practice this is not the case, so absorption is not a desired property. In contracts the choice depends on the subjects that execute the actions and it is not always that they will choose the least demanding action. The criteria for choosing the actions may depend on the subject performing the action and may also depend on which actions are in the choice.

Definition 2.3 (conflict relation)**.** *We consider a symmetric and irreflexive relation over the basic actions $\mathcal{A}_B$, which we call conflict relation and denote by $\#_\mathcal{C} \subseteq \mathcal{A}_B \times \mathcal{A}_B$.*

The *conflict relation* is a notion often found in legal contracts and is given a priori. The intuition of the conflict relation is that if two actions are in conflict then the actions cannot be done at the same time. This intuition explains the need for the following equational implication:

$$(18) \quad a \,\#_{\mathcal{C}}\, b \rightarrow a \times b = \mathbf{0} \qquad \forall a, b \in \mathcal{A}_B.$$

There is *no transitivity* of $\#_{\mathcal{C}}$ which is natural as action "drive" may be in conflict with both "drink" and "talk at the phone" but still one can "drink and talk at the phone" at the same time.

Definition 2.4 (canonical form)**.** *We say that an action α is in* canonical form *denoted by $\underline{\alpha}$ iff it has the following form:*

$$\underline{\alpha} = \underset{i \in I}{+} \; \alpha^i_\times \cdot \underline{\alpha^i}$$

where $\alpha^i_\times \in \mathcal{A}^\times_B$ and $\underline{\alpha^i} \in \mathcal{A}$ is an action in canonical form. The indexing set I is finite as $\alpha^i_\times \in \mathcal{A}^\times_B$ are finite; therefore there is a finite number of application of the $+$ combinator. Actions $\mathbf{0}$ and $\mathbf{1}$ are in canonical form.

Theorem 2.1. *For every action α there exists a corresponding action $\underline{\alpha}$ in canonical form and equivalent to α.*

For the deontic modalities, one important notion is *action negation*. There have been a few works related to negation of actions for PDL-like logics [Mey88, HKT00, LW04, Bro03]. In [Mey88], the same as in [HKT00] action negation is with respect to the universal relation which for PDL gives undecidability. Decidability of PDL with negation of only atomic actions has been achieved in [LW04]. A so called "relativized action complement" is defined in [Bro03] which is the complement of an action (not w.r.t. the universal relation but) w.r.t. a set formed of atomic actions closed under the application of the action combinators. This kind of negation still gives undecidability when several action combinators are involved. Any deontic logic of actions that is based on PDL and considers action negation as above is bound to be undecidable.

In our view action negation encodes the violation of an obligation. Intuitively, action negation says that the negation $\overline{\alpha}$ of action α is the action given by all the immediate actions that *take us outside* the tree of α [BWM01]. With $\underline{\alpha}$ it is easy to formally define $\overline{\alpha}$.

Definition 2.5 (action negation)**.** *The* action negation *is denoted by $\overline{\alpha}$ and is defined as a function $\overline{} : \mathcal{A} \rightarrow \mathcal{A}$ (i.e. action negation is not a principal combinator for the actions) and works on the equivalent canonical form $\underline{\alpha}$ as:*

$$\overline{\alpha} = \overline{\underset{i \in I}{+} \; \alpha^i_\times \cdot \underline{\alpha^i}} = \underset{\beta_\times \in \overline{R}}{+} \beta_\times \; + \; \underset{j \in J}{+} \; \gamma^j_\times \cdot \overline{\underset{i \in I'}{+} \underline{\alpha^i}}$$

Consider $R = \{\alpha^i_\times \mid i \in I\}$. The set $\overline{R}$ contains all the concurrent actions $\beta_\times$ with the property that $\beta_\times$ is not more demanding than any of the actions $\alpha^i_\times$:

$$\overline{R} = \{\beta_\times \mid \beta_\times \in \mathcal{A}^\times_B \text{ and } \forall i \in I, \alpha^i_\times \not\leq_\times \beta_\times\};$$

and $\gamma^j_\times \in \mathcal{A}^\times_B$ and $\exists \alpha^i_\times \in R$ s.t. $\alpha^i_\times \leq_\times \gamma^j_\times$. The indexing set $I' \subseteq I$ is defined for each $j \in J$ as:

$$I' = \{i \in I \mid \alpha^i_\times \leq_\times \gamma^j_\times\}.$$

Remark: We elaborate here on a discussion we had [PS07c] about the nature of action negation. Literaly one may consider two kinds of action negation: one "anything else but a" and another "not doing a". We choose the first type as in our setting the second type of action negation is not found.

The negation operation formalizes the fact that an action is not performed. In an eager system this boils down to performing the action given by $\overline{\alpha}$. In other words *not performing* action α means either not performing any of its immediate actions $\alpha_\times^i$, or by performing one of the immediate actions and then not performing the remaining action. Note that to perform an action $\alpha_\times^i$ means to perform any action that includes $\alpha_\times^i$. Therefore in the negation we may have actions $\gamma_\times^j$ which include more immediate actions, e.g. $\alpha = a \cdot b + c \cdot d$ and may perform $\gamma_\times^j = a \times c$. At this point we need to look at both actions b and d in order to derive the negation, e.g. performing now d means that α was done, whereas performing c means that α was not done (and $a \times c \cdot c$ must be part of negation). Note that because $\mathcal{A}_B^\times$ is finite then we have finite summation in the action negation; the maximum number of summands being at most $|\mathcal{A}_B^\times|$.

It was shown in [Pri08b] that action negation is not independent of the representation. This means that applied to two equivalent actions the negation may yield not equivalent actions. As noted in [Pri08b], the problem comes from the fact that the canonical form on which the action negation is applied is not unique. The system of equations of Table 2 can be associated a terminating and confluent term rewriting system. Therefore, there exists a unique normal form for actions, but we cannot pin-point it syntactically. For the present work we are interested in a slightly weaker notion of *almost normal form*.

Definition 2.6 (almost normal form)**.** *We call* almost normal form*, and denote it* $\underline{\alpha}!$*, an action to which only axiom (8) can be applied (and none other of the axioms of Table 2).*

To apply an axiom means to apply the rule obtained from directing the axiom from left to right; see [BN98] for details on rewriting theory. The next two theorems relate the almost normal form with the canonical form and with the normal form of an action.

We prove that the canonical form which is also an almost normal form is *unique*. Moreover, we know its syntactic form: it is a canonical form with the restrictions as in Theorem 2.2. The most important restrictions are c and d which take care of the special actions $\mathbf{1}$ and $\mathbf{0}$. We call a canonical form which is also an almost normal form a *canonical almost normal form* and we abbreviate it by c.a.n.f.. The action negation applied as in the Definition 2.5 but on a c.a.n.f. is guaranteed to return a unique result.

Theorem 2.2. *A canonical form* $\underline{\alpha} = +_{i \in I}\ \alpha_\times^i \cdot \underline{\alpha^i}$ *is an almost normal form iff it respects the following restrictions:*

a. $\forall i \in I, \forall a \in \mathcal{A}_B$ *then* a *appears at most once in* $\alpha_\times^i$*;*

b. $\forall i, j \in I,\ \alpha_\times^i \neq \alpha_\times^j$*;*

c. $\forall i \in I$*, either*

c.1. $\underline{\alpha^i} \in \mathcal{A} \setminus \{\mathbf{0}, \mathbf{1}\}$ *or*

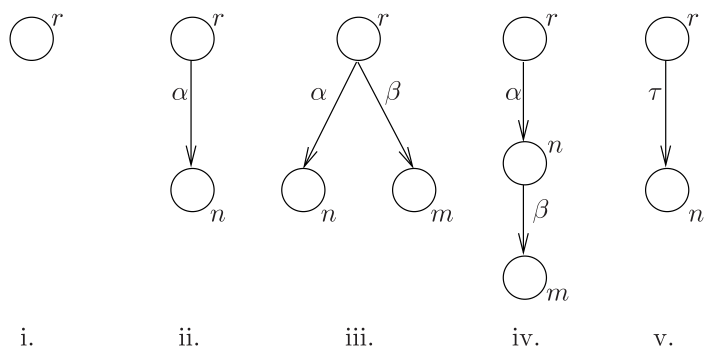

Figure 1: Examples of finite rooted trees with labeled edges.

c.2. $\underline{\alpha^i}$ *does not exist and in this case* $\alpha^i_\times \in \mathcal{A}^\times_B \cup \{\mathbf{1}\}$ *is allowed to be also* $\mathbf{1}$*;*

d. $\underline{\alpha^i}$ *respects strictly the restrictions a, b, and c.*

Theorem 2.3. *The normal form of an action* α *is obtained from the almost normal form* $\underline{\alpha!}$ *by a finite application of axioms (8) and (6).*

Corollary 2.4. *The c.a.n.f. of an action* α *is* unique *(modulo associativity and commutativity).*

The corollary is immediate from the proof of Theorems 2.2 and 2.3. There is no application of the axioms to a c.a.n.f. which yields another c.a.n.f..

Definition 2.7 (rooted tree)**.** *A* rooted tree *is an acyclic connected graph* $(\mathcal{N}, \mathcal{E})$ *with a designated node* r *called* root *node.* $\mathcal{N}$ *is the set of* nodes *and* $\mathcal{E}$ *is the set of* edges *(where an edge is an ordered pair of nodes* (n,m)*). We consider* rooted trees with labeled edges *and denote the labeled directed edges with* (n,α,m) *and the tree with* $(\mathcal{N}, \mathcal{E}, \mathcal{A}_B)$*. The labels* $\alpha \in 2^{\mathcal{A}_B}$ *are* sets of basic labels*; e.g.* $\alpha_1 = \{a,b\}$ *or* $\alpha_2 = \{a\}$ *with* $a,b \in \mathcal{A}_B$*. Labels are compared for set equality (or set inclusion). Note the special* empty set *label. We consider a special label* Λ *to stand for an impossible label. We restrict our presentation to* finite *rooted trees (i.e., there is no infinite path in the graph starting from the root node). The set of all such defined trees is denoted* $\mathcal{T}$*.*

Notation: When the label of an edge is not important (i.e. it can be any label) we may use the notation (n,m) instead of (n,α,m) $\forall \alpha \in 2^{\mathcal{A}_B}$. All nodes $\{m \,|\, (n,m)\}$ are called the *child* nodes of n. A path which cannot be extended with a new transition is called *final*. The final nodes on each final path are called *leaf nodes*; denote by $\mathit{leafs}(T) = \{n \mid n \text{ is a leaf node}\}$.

Example: Rooted trees with labeled edges are given in Figure 1:

i. $(\{r\}, \emptyset, \emptyset)$ - the tree with only one node the root, and no edges;

ii. $(\{r,n\}, \{(r,\alpha,n)\}, \{\alpha\})$ - the tree with only one edge;

iii. $(\{r,n,m\}, \{(r,\alpha,n),(r,\beta,m)\}, \{\alpha,\beta\})$ - the tree with two edges coming from the root r;

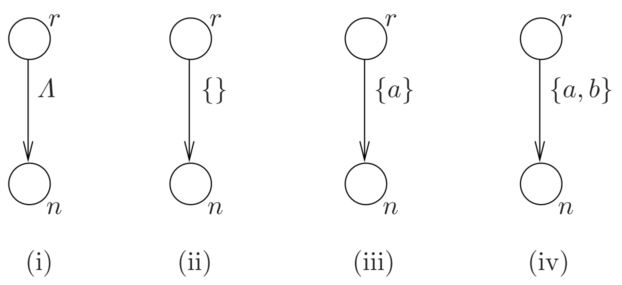

Figure 2: Trees corresponding to $\mathbf{0}$, $\mathbf{1}$, $a \in \mathcal{A}_B$, and $a \times b \in \mathcal{A}_B^\times$.

iv. $(\{r, n, m\}, \{(r, \alpha, n), (n, \alpha, m)\}, \{\alpha, \beta\})$ - the tree with only one path of two edges;

v. $(\{r, n\}, \{(r, \tau, n)\}, \{\tau\})$ - the tree with only one edge labeled by the empty label τ.

Definition 2.8 (tree isomorphism)**.** *$T_1 = (\mathcal{N}_1, \mathcal{E}_1, \mathcal{A}_1)$ and $T_2 = (\mathcal{N}_2, \mathcal{E}_2, \mathcal{A}_2)$ are called isomorphic, denoted $T_1 \doteq T_2$, iff $\mathcal{A}_1 = \mathcal{A}_2$, and there is a bijection $rn : \mathcal{N}_1 \to \mathcal{N}_2$ s.t. $rn(root_1) = root_2$ and $\forall (n, \alpha, m) \in \mathcal{E}_1$ then $(rn(n), \alpha, rn(m)) \in \mathcal{E}_2$.*

Theorem 2.5 (interpretation of actions)**.** *For any action α there exists a tree representation corresponding to the canonical almost normal form $\underline{\alpha}!$.*

Proof. The *representation* is an interpretation function $I_{\mathcal{CA}} : \mathcal{A} \to \mathcal{T}$ which interprets all action terms as trees. More precisely, given an arbitrary action of $\mathcal{A}$, the c.a.n.f. is computed first and then $I_{\mathcal{CA}}$ generates the tree representation of the canonical form. We do not give an algorithm for computing the c.a.n.f. as one may simply apply exhaustively the axioms excluding (8)

The function $I_{\mathcal{CA}}$ is defined inductively. The basis is to interpret each concurrent action of $\mathcal{A}_B^\times$ as a tree with labeled edges from $2^{\mathcal{A}_B}$ as pictured in Fig.2. Note that actions of $\mathcal{A}_B^\times \cup \{\mathbf{0}, \mathbf{1}\}$ are in canonical form. For a general action in canonical form $\underline{\alpha} = +_{i \in I}\ \alpha_\times^i \cdot \underline{\alpha^i}$ the tree is generated by adding one branch to the root node for each element $\alpha_\times^i$ of the top summation operation. The label of the branch is the set $\{\alpha_\times^i\}$ corresponding to the concurrent action. The construction continues inductively by attaching at the end of each newly added branch the tree interpretation of the smaller action $\underline{\alpha^i}$. Intuitively, $+$ provides the branching in the tree, and $\cdot$ provides the parent-child relation on each branch. $\square$

Henceforth we work only with the c.a.n.f. because it is unique. We also work with the unique tree representation $I_{\mathcal{CA}}(\alpha)$. Because of the completeness of the axiom system of Table 2 w.r.t. the tree interpretation cf. [Pri08b], we have that $\alpha = \beta$ iff $I_{\mathcal{CA}}(\alpha) \doteq I_{\mathcal{CA}}(\beta)$. In other words, if the trees interpreting two actions are isomorphic (i.e. denote the same tree) then the actions are equal. Therefore we can work with the trees or the actions interchangeably.

2.3 Dynamic Actions

The actions inside the dynamic box modality are the regular actions of PDL enriched with the synchrony operator. A study of the formalisation and properties of these actions was done in [Pri08b] under the name of *synchronous Kleene algebra with tests*. In this section we summarize some of these results because they are required for giving the smantics of $\mathcal{CL}$ in Section 3.

Definition 2.9. *The dynamic actions are given by* $(\mathcal{A}, \mathcal{A}^?, +, \cdot, \times, ^*, \neg, \mathbf{0}, \mathbf{1})$ *which is an order sorted algebraic structure with* $\mathcal{A}^? \subseteq \mathcal{A}$ *which combines the previously defined deontic actions with a Boolean algebra in a special way we see in this section. Moreover, the repetition operation* * *is added. The structures* $(\mathcal{A}^?, +, \cdot, \neg, \mathbf{0}, \mathbf{1})$ *and* $(\mathcal{A}^?, +, \times, \neg, \mathbf{0}, \mathbf{1})$ *are Boolean algebras and the Boolean negation operator* $\neg$ *is defined only on Boolean elements of* $\mathcal{A}^?$*.*

Notation: The elements of the set $\mathcal{A}^?$ are called *tests* (or *guards*) and are included in the set of actions of the $\mathcal{SKA}$ algebra (i.e. tests are special actions; $\mathcal{A}^? \subseteq \mathcal{A}$). As in the case of actions, the Boolean algebra is generated by a finite set $\mathcal{A}^?_B$ of *basic tests* which can be thought of as assertions. We denote tests by $\varphi, \phi, \ldots$ and basic tests by $p, q, \ldots$.[4] Note the overloading of the functional symbols $+, \cdot, \times$, and functional constants $\mathbf{0}, \mathbf{1}$: over arbitrary actions they have the meaning as in the previous section; whereas, over tests they take the meaning of the classical disjunction (for $+$, which sometimes we denote by $\vee$), conjunction (for $\cdot$ and $\times$ which sometimes we denote by $\wedge$), falsity and truth (for $\mathbf{0}, \mathbf{1}$ which we often denote by $\bot$ and $\top$). In this richer context the elements of $\mathcal{A}$ (i.e. the actions and tests) are the syntactic terms constructed with the grammar below:

$$\begin{array}{lcll} \beta & ::= & a \mid \varphi \mid \beta + \beta \mid \beta \cdot \beta \mid \beta \times \beta \mid \beta^* & \qquad \textit{actions} \\ \varphi & ::= & p \mid \mathbf{0} \mid \mathbf{1} \mid \varphi + \varphi \mid \varphi \cdot \varphi \mid \varphi \times \varphi \mid \neg \varphi & \qquad \textit{tests} \end{array}$$

The intuition behind tests is that for an action $\varphi \cdot \beta$ it is the case that action β can be performed only if the test succeeds (the condition φ is satisfied). We do not go into details about the properties of a Boolean algebra as these are classical results. For a thorough understanding see [Koz97] and references therein.

It is natural to think of $\mathbf{1}$ as $\top$ because testing a tautology always succeeds; i.e. $\top \cdot \beta = \mathbf{1} \cdot \beta = \beta$, which says that the action β can always be performed after a $\top$ test. The dual is $\bot = \mathbf{0}$ meaning that testing a falsity never succeeds, and thus, any following action β is never performed; i.e. $\mathbf{0} \cdot \beta = \mathbf{0} = \bot \cdot \beta = \bot$ (the action sequence stops when it reaches the $\bot$ test).

Definition 2.10 (extra axiom)**.** *We give the equivalent of axiom (17) for tests:*

$$\textit{(17')} \quad (\varphi \cdot \beta) \times (\varphi' \cdot \beta') = (\varphi \times \varphi') \cdot (\beta \times \beta') \quad \forall \varphi, \varphi' \in \mathcal{A}^?.$$

Note that $\top \in \mathcal{A}^?$ and therefore this axiom allowes sequences of actions with $\mathbf{1}$, which was not the case in axiom (17). On the other hand $\mathbf{1}$ is dealt with only in conjunction with another test, and not with another action. Particular instances of this axiom are $\beta \times \varphi = \varphi \cdot \beta$ and $\varphi \times \beta = \varphi \cdot \beta$. Note that $(\varphi \cdot \beta) \times (\varphi' \cdot \beta')$ is $(\varphi \wedge \varphi') \cdot \beta \times \beta'$.

Each dynamic action denotes a set of *guarded concurrent strings.*

[4] Note that in this section we simplify the notation and drop the ? for the tests. Nevertheless, in the logic discourse of $\mathcal{CL}$ we use the ? notation.

Definition 2.11 (guarded concurrent strings)**.** *Over the set of basic tests $\mathcal{A}_B^?$ we define* atoms *as functions $\nu : \mathcal{A}_B^? \to \{0,1\}$ assigning a Boolean value to each basic test. Consider the finite alphabet $\mathcal{P}(\mathcal{A}_B)$ of all the subsets of basic actions (denoted by x, y). A* guarded concurrent string *(denoted by u, v, w) is a sequence*

$$w = \nu_0 x_1 \nu_1 \dots x_n \nu_n, \qquad n \geq 0,$$

where ν_i are atoms and $x_i \in \mathcal{P}(\mathcal{A}_B)$ are sets of basic actions. We define $first(w) = \nu_0$ and $last(w) = \nu_n$. We define two mappings over guarded concurrent strings: τ which returns the associated (unguarded) concurrent string; i.e. $\tau(w) = x_1 \dots x_n$ and π which returns the sequence of guards; i.e. $\pi(w) = \nu_0 \nu_1 \dots \nu_n$.

Similar to what we did for the deontic actions we can construct for each action β an automaton which accepts all and only the guarded concurrent strings denoted by β. The completeness results of [Pri08b] ensures that working with the dynamic actions or with the automata is the same (they are interchangeable).

Proposition 2.6 (automata for guards)**.** *For the set of Atoms there exists a class of finite state automata which accept all and only the subsets of atoms.*

Proof. This is a folk result. Recall that the set *Atoms* is the finite set of functions $\nu : \mathcal{A}_B^? \to \{0,1\}$ from the finite domain of basic tests $\mathcal{A}_B^?$ to the truth set $\{0,1\}$. We define a class of finite automata which recognize functions $f : A \to B$ with finite domain and codomain, and thus, the languages accepted are subsets of the finite set of all possible functions f and nothing else. As a particular case when $A = \mathcal{A}_B^?$ and $B = \{0,1\}$ this class of finite automata will accept languages of atoms.

Take a finite state automaton $M = (S, \Sigma, S_0, \rho, F)$ with the following restrictions:

a. $S_0 \subseteq S$, $F \subseteq S$ the set of initial and final states;

b. $\Sigma = A \cup B$ the two sorted (disjoint) alphabet;

c. $\rho = \rho_A \cup \rho_B$ the transition relation as the union of the two disjoint relations $\rho_A \subseteq S \times A \times S$ and $\rho_B \subseteq S \times B \times S$;

d. Any final trace (i.e. trace which starts in a start state and ends in a final state) of the automaton is of length $2 * |A|$ and no element of A appears twice on a final trace (i.e. no two transitions in the final trace are labeled with the same $a \in A$);

e. $\forall (s_1, b, s_2) \in \rho_B$ then $s_1 \notin S$ and $\nexists (s_2, b', s_3) \in \rho_B$;

f. $\forall (s_1, a, s_2) \in \rho_A$ then $s_2 \notin F$ and $\nexists (s_2, a', s_3) \in \rho_A$.

Denote the set of all such automata by $\mathcal{M}$. □

Corollary 2.7. *Automata as defined in Proposition 2.6 but on the particular alphabet $\Sigma = \mathcal{A}_B^? \cup \{0,1\}$ accept all and only the sets of atoms.*

Corollary 2.8. *Automata of Proposition 2.6 are closed under the classical operations on finite automata union (denoted $\cup$) and intersection (denoted $\cap$); and correspond respectively to union of sets of functions and intersection of sets of functions. Automata of Proposition 2.6 are not closed under concatenation.*

Definition 2.12 (automata on guarded concurrent strings)**.** *Consider a two level finite automaton $N = (S, \mathcal{P}(\mathcal{A}_B), S_0, \rho, F, \lceil\cdot\rceil)$. It consists at the first level of a finite automaton on concurrent strings $(S, \mathcal{P}(\mathcal{A}_B), S_0, \rho, F)$, together with a map $\lceil\cdot\rceil : S \to \mathcal{M}$. An automaton on concurrent strings (i.e. the first level automation) consists of a finite set of* states *S, the finite* alphabet of concurrent actions *$\mathcal{P}(\mathcal{A}_B)$ (i.e. the powerset of the set of basic actions $\mathcal{A}_B$), a set of* initial designated states *$S_0 \subseteq S$, a* transition relation *$\rho : \mathcal{P}(\mathcal{A}_B) \to S \times S$, and a set of final states F. The mapping associates with each state of the first level an automaton $M \in \mathcal{M}$ as defined in Proposition 2.6 which accepts atoms. The automata in the states make the second (lower) level. Denote the language of atoms accepted by $\lceil s\rceil$ with $\mathcal{L}(\lceil s\rceil)$.*

Definition 2.13 (acceptance)**.** *A* run *of the automaton is a sequence of transitions starting in the initial state, i.e. $s_0 \xrightarrow{\{\alpha_\times^1\}} s_1 \xrightarrow{\{\alpha_\times^2\}} s_2 \ldots \xrightarrow{\{\alpha_\times^n\}} s_n$. A run is called* accepting *if it ends in a final state, i.e. $s_n \in F$. We say that* a guarded concurrent string w is accepted by an automaton N *iff there exists an accepting run of the first level automaton which accepts $\tau(w)$ and for each state s_i of the run there exists an accepting run of the automaton $\lceil s_i\rceil$ which accepts the corresponding atom ν_i of w.*

It is easy to see that the automata on guarded concurrent strings can be considered as ordinary finite state automata. The two level definition is useful for the definitio of the *fusion product* and *concurrent composition* operations over these automata (see [Pri08b]). Henceforth we denote automata on guarded concurrent strings (as defined in Definition 2.12) by GNFA.

3 Direct semantics of $\mathcal{CL}$

In [PS07b] we have given an interpretation for $\mathcal{CL}$ with the help of a translation function into $\mathcal{C}\mu$, an extension of μ-calculus which we introduced for this purpose. In this section we give two direct semantics for the $\mathcal{CL}$ language using the algebraic notions for the actions presented in sections 2.2 and 2.3.

First we give a branching semantics in terms of *normative structures*. This semantics is intended for model checking and for reasoning about the contracts written in $\mathcal{CL}$.

The second semantics is a linear semantics in terms of respecting traces of actions. It is more poor than the branching semantics, as the structures (i.e. action traces) cannot carry all the information that a normative structure carries. The purpose for the linear semantics is to do run-time monitoring of contracts written in $\mathcal{CL}$. The contents about the trace semantics was presented in [KPS08a] and here we only remind the main notions. We need these in order to give the relation between the two semantics which was not done before.

3.1 Branching semantics in terms of normative structures

The formulas $\mathcal{C}$ of the logic are given a model theoretic semantics in terms of (what we define as) *normative structures.*

Definition 3.1 (normative structure)**.** *A* labeled Kripke structure *is a structure $K = (\mathcal{W}, \rho, \mathcal{V})$ where $\mathcal{W}$ is a set of* worlds *(states), $\mathcal{V} : \mathcal{W} \to 2^{\Phi_B}$ is a* valuation function *returning for each world the set of propositional constants which hold in that world. $\mathcal{A}_B$ is a finite set of* basic labels*, $2^{\mathcal{A}_B}$ is the powerset representing the* labels *of the structure, and $\rho : 2^{\mathcal{A}_B} \to 2^{\mathcal{W} \times \mathcal{W}}$ is a function returning for each label a relation on the set of worlds. In a* deterministic *labeled Kripke structure the function ρ associates to each label a partial function instead of a relation, therefore for each label from one world there is at most one reachable world. A* normative structure *is a deterministic labeled Kripke structure with a marking function $\varrho : \mathcal{W} \to 2^{\Psi}$ which marks each state with one or several markers from $\Psi = \{\circ_a, \bullet_a \mid a \in \mathcal{A}_B\}$. The marking function respects the restriction that no state can be marked by both $\circ_a$ and $\bullet_a$ (for any $a \in \mathcal{A}_B$). Normative structures are denoted $K^{\mathcal{N}} = (\mathcal{W}, \rho, \mathcal{V}, \varrho)$. A* pointed normative structure *is a normative structure with a designated state i (denoted by $K^{\mathcal{N}}, i$).*

Notation: We denote by an indexed t a node of a tree (or by r the root) and by an indexed s (or i for initial) a state of a normative structure. Henceforth we use the more graphical notation $t \xrightarrow{\alpha} t'$ ($s \xrightarrow{\alpha} s'$) for an edge (transition) in a tree (normative structure) instead of the classical one (t, α, t') ($(s, s') \in \rho(\alpha)$) that we had before. Note that we consider both the trees and the normative structures to have the same set of basic labels $\mathcal{A}_B$. For the sake of notation we can view the valuation of one particular state $\mathcal{V}(s)$ as a function from Φ_B to $\{0, 1\}$ where $\forall \varphi \in \Phi_B$, $\mathcal{V}(s)(\varphi) = 1$ iff $\varphi \in \mathcal{V}(s)$ and $\mathcal{V}(s)(\varphi) = 0$ iff $\varphi \notin \mathcal{V}(s)$. Therefore we can say that $\mathcal{V}(s)$ is accepted by the automata of the second level and write $\mathcal{V}(s) \in \mathcal{L}(\lceil s \rceil)$.

The marking function and the markers are needed to identify obligatory (i.e. $\circ$) and prohibited (i.e. $\bullet$) actions. Markers with different purposes were used in [Mey88] to identify violations of obligations, in [vdM96] to mark permitted transitions, and in [CM07] to identify permitted events.

Definition 3.2 (simulation)**.** *For a tree $T = (\mathcal{N}, \mathcal{E}, \mathcal{A}_B)$ and a normative structure $K^{\mathcal{N}} = (\mathcal{W}, \rho, \mathcal{V}, \varrho)$ we define a relation $\mathcal{S} \subseteq \mathcal{N} \times \mathcal{W}$ which we call the* simulation *of the tree node by the state of the structure.*

$$t \,\mathcal{S}\, s \text{ iff } \forall t \xrightarrow{\gamma} t' \in T,\ \exists s \xrightarrow{\gamma'} s' \in K^{\mathcal{N}} \text{ s.t. } \gamma \subseteq \gamma' \wedge t' \mathcal{S} s' \text{ and}$$
$$\forall s \xrightarrow{\gamma'} s' \in K^{\mathcal{N}} \text{ with } \gamma \subseteq \gamma' \text{ then } t' \,\mathcal{S}\, s'.$$

We say that a tree T, with root r is simulated by a normative structure $K^{\mathcal{N}}$ w.r.t. a state s, denoted $T \,\mathcal{S}_s\, K^{\mathcal{N}}$, iff $r \,\mathcal{S}\, s$.

Note two differences with the classical definition of simulation: first, the labels of the normative structure may be bigger than the labels in the tree because respecting an obligatory action means executing an action which includes it (is bigger). We can drop this condition and consider only $\gamma = \gamma'$, in which case we call the relation *strong simulation* and denote by $\mathcal{S}^s$. Second, any transition in the normative structure that can simulate an edge in the tree must enter under the simulation relation. This is because from the state s' onwards we need to be able to continue to look in the structure for the remaining tree (to see that it is simulated). We can weaken the definition by combining this condition with the one before into: $\forall t \xrightarrow{\gamma} t' \in T$, $\forall s \xrightarrow{\gamma'} s' \in K^{\mathcal{N}}$ with $\gamma \subseteq \gamma'$ then $t' \tilde{\mathcal{S}}\, s'$. We call the resulting relation *partial simulation* and denote it by $\tilde{\mathcal{S}}$.

Definition 3.3 (maximal simulating structure)**.** *Whenever $T\, \mathcal{S}_i\, K^{\mathcal{N}}$ then we call the* maximal simulating structure w.r.t. T and i*, and denote it by* $K^{T,i}_{max} = (\mathcal{W}', \rho', \mathcal{V}', \varrho')$ *the sub-structure of* $K^{\mathcal{N}} = (\mathcal{W}, \rho, \mathcal{V}, \varrho)$ *s.t.:*

a. $i \in \mathcal{W}'$

b. $\mathcal{V}' = \mathcal{V}|_{\mathcal{W}'}$ *and* $\varrho' = \varrho|_{\mathcal{W}'}$

c. $\forall t \xrightarrow{\gamma} t' \in T$ *then* $\forall s \xrightarrow{\gamma'} s' \in K^{\mathcal{N}}$ *s.t.* $t\, \mathcal{S}\, s \wedge \gamma \subseteq \gamma' \wedge t'\, \mathcal{S}\, s'$ *do add* s' *to* $\mathcal{W}'$ *and add* $s \xrightarrow{\gamma'} s'$ *to* ρ'.

We call the non-simulating remainder of $K^{\mathcal{N}}$ w.r.t. T and i *the sub-structure* $K^{T,i}_{rem} = (\mathcal{W}'', \rho'', \mathcal{V}'', \varrho'')$ *of* $K^{\mathcal{N}}$ *s.t.:* $s \xrightarrow{\gamma} s' \in \rho''$ *iff* $s \xrightarrow{\gamma} s' \notin K^{T,i}_{max} \wedge s \in K^{T,i}_{max} \wedge \exists s \xrightarrow{\gamma} s'' \in K^{T,i}_{max}$*; and* $s \in \mathcal{W}''$ *iff* s *is part of a transition in* $K^{T,i}_{rem}$*; and* $\mathcal{V}'' = \mathcal{V}|_{\mathcal{W}''}$ *and* $\varrho'' = \varrho|_{\mathcal{W}''}$.

Definition 3.4 (semantics)**.** *We give in Table 4 a recursive definition of the* satisfaction relation $\models$ *of a formula* $\mathcal{C}$ *w.r.t. a pointed normative structure* $K^{\mathcal{N}}, i$*; it is written* $K^{\mathcal{N}}, i \models \mathcal{C}$ *and is read as "*$\mathcal{C}$ *is* satisfied *in the normative structure* $K^{\mathcal{N}}$ *at state* i*". We write* $K^{\mathcal{N}}, i \not\models \mathcal{C}$ *whenever* $\models$ *is not the case. We say that "*$\mathcal{C}$ *is* globally satisfied *in* $K^{\mathcal{N}}$*", and write* $K^{\mathcal{N}} \models \mathcal{C}$ *iff* $\forall s \in K^{\mathcal{N}}$, $K^{\mathcal{N}}, s \models \mathcal{C}$*. A formula is* satisfiable *iff* $\exists K^{\mathcal{N}}, \exists s \in K^{\mathcal{N}}$ *s.t.* $K^{\mathcal{N}}, s \models \mathcal{C}$*. A formula is* valid *(denoted* $\models \mathcal{C}$*) iff* $\forall K^{\mathcal{N}}, K^{\mathcal{N}} \models \mathcal{C}$.

The propositional connectives have the classical semantics. More interesting and particular to our logic is the interpretation of the deontic modalities. For the $O_{\mathcal{C}}$ the semantics has basically two parts: the second part is just the last line and states that if the obligation is violated (i.e. $\overline{\alpha}$ negation of the action is performed) then the reparation $\mathcal{C}$ should hold. This definition is similar to the definition of the box modality of PDL only that here it is applied to the negation of the action (i.e. it looks only at the leafs and it uses strong simulation to have exactly the same labels as the negation action tree). The first part of the semantics is the interpretation of the obligation. The first line says how to walk on the structure depending on the tree of the action α. The simulation relation is used because in the structure there may be transitions labeled with actions that are greater which intuitively if we do these actions then the obligation of α is still respected. The simulation relation also takes care that all the choices of an action appear as transitions in the structure. Lines second-third mark all the transitions (their ending states) of the structure which simulate edges in the tree with markers $\circ_a$ corresponding to the labels of the simulated edge. This is needed both for the proof of the main property $O_{\mathcal{C}}(\alpha) \wedge O_{\mathcal{C}}(\beta) \rightarrow O_{\mathcal{C}}(\alpha \times \beta)$ and also in proving $O_{\mathcal{C}}(\alpha) \rightarrow \neg F_{\mathcal{C}}(\alpha)$ which relates obligations and prohibitions. Lines four-five ensure that no other reachable relevant transitions of the structure (i.e. from the non-simulating remainder structure) are marked with obligation markers $\circ$.

For the $F_{\mathcal{C}}$ modality we use partial simulation $\tilde{\mathcal{S}}_i$ in order to have our intuition that if an action is not present as a label of an outgoing transition of

$$
\begin{array}{ll}
K^{\mathcal{N}}, i \models \varphi & \text{iff } \varphi \in \mathcal{V}(i). \\
K^{\mathcal{N}}, i \not\models \bot & \\
K^{\mathcal{N}}, i \models \mathcal{C}_1 \rightarrow \mathcal{C}_2 & \text{iff whenever } K^{\mathcal{N}}, i \models \mathcal{C}_1 \text{ then } K^{\mathcal{N}}, i \models \mathcal{C}_2. \\
K^{\mathcal{N}}, i \models O_{\mathcal{C}}(\alpha) & \text{iff } I_{\mathcal{CA}}(\alpha)\ \mathcal{S}_i\, K^{\mathcal{N}}, \quad \text{and} \\
 & \quad \forall t \xrightarrow{\gamma} t' \in I_{\mathcal{CA}}(\alpha),\ \forall s \xrightarrow{\gamma'} s' \in K^{\mathcal{N}} \text{ s.t. } t\, \mathcal{S}\, s \wedge \gamma \subseteq \gamma' \\
 & \qquad \text{then } \forall a \in \mathcal{A}_B \text{ if } a \in \gamma \text{ then } \circ_a \in \varrho(s'), \quad \text{and} \\
 & \quad \forall s \xrightarrow{\gamma'} s' \in K_{rem}^{I_{\mathcal{CA}}(\alpha), i} \text{ then } \forall a \in \mathcal{A}_B \text{ if } a \in \gamma' \text{ then } \circ_a \notin \varrho(s'), \quad \text{and} \\
 & \quad K^{\mathcal{N}}, s \models \mathcal{C} \quad \forall s \in N \text{ with } t\, \mathcal{S}^s\, s \wedge t \in \mathit{leafs}(I_{\mathcal{CA}}(\overline{\alpha})). \\
K^{\mathcal{N}}, i \models F_{\mathcal{C}}(\alpha) & \text{iff } I_{\mathcal{CA}}(\alpha)\ \tilde{\mathcal{S}}_i\, K^{\mathcal{N}} \text{ then} \\
 & \quad \forall \sigma \in I_{\mathcal{CA}}(\alpha) \text{ a full path s.t. } \sigma\ \mathcal{S}_i\, K^{\mathcal{N}}, \\
 & \quad \exists t \xrightarrow{\gamma} t' \in \sigma \text{ s.t. } \forall s \xrightarrow{\gamma'} s' \in K^{\mathcal{N}} \text{ with } t\, \mathcal{S}\, s \wedge \gamma \subseteq \gamma' \text{ then} \\
 & \qquad \forall a \in \mathcal{A}_B \text{ if } a \in \gamma' \text{ then } \bullet_a \in \varrho(s') \quad \text{and} \\
 & \quad K^{\mathcal{N}}, s \models \mathcal{C} \quad \forall s \in N \text{ with } t\, \mathcal{S}\, s \wedge t \in \mathit{leafs}(\sigma). \\
K^{\mathcal{N}}, i \models P(\alpha) & \text{iff } I_{\mathcal{CA}}(\alpha)\ \mathcal{S}_i\, K^{\mathcal{N}}, \text{ and} \\
 & \quad \forall t \xrightarrow{\gamma} t' \in I_{\mathcal{CA}}(\alpha),\ \forall s \xrightarrow{\gamma} s' \in K^{\mathcal{N}} \text{ s.t. } t\, \mathcal{S}^s\, s \wedge t'\, \mathcal{S}^s\, s' \\
 & \qquad \text{then } \forall a \in \mathcal{A}_B \text{ if } a \in \gamma \text{ then } \bullet_a \notin \varrho(s'). \\
K^{\mathcal{N}}, i \models [\beta]\mathcal{C} & \text{iff } \forall t \in \mathcal{W} \text{ with } (i,t) \in \rho(\beta) \text{ then } K^{\mathcal{N}}, t \models \mathcal{C}
\end{array}
$$

$$
\begin{array}{rl}
\rho(\beta) = & \{(s,t) \mid \exists k, \exists \sigma = x_0 \ldots x_k \text{ a final path in } GNFA(\beta), \\
 & \quad \exists s_0 \ldots s_k \in \mathcal{W} \text{ with } s_0 = s \text{ and } s_k = t, \text{ and} \\
 & \quad \forall 0 \leq i \leq k,\ \mathcal{V}(s_i) \in \mathcal{L}(\lceil x_i \rceil), \text{ and} \\
 & \quad \forall 0 \leq i < k \text{ with } x_i \xrightarrow{\beta_\times} x_{i+1} \in \sigma \text{ then } s_i \xrightarrow{\beta_\times} s_{i+1} \in K^{\mathcal{N}}\}
\end{array}
$$

Table 3: Semantics for $\mathcal{CL}$.

the model then the action is *by default* considered forbidden. In the second line we consider *all* paths in order to respect the intuition that $F(a+b) = F(a) \wedge F(b)$, prohibition of a choice must prohibit all. In the third line we consider just the *existence* of an edge on each full path in order to respect the intuition that forbidding a sequence means forbidding some action on that sequence. Note that we are interested only in *full* paths simulated by the structure because for the other paths some of the transitions are missing in the structure and thus there is some action on the sequence which is forbidden. For a chosen edge we look for all the transitions of the normative structure from the chosen node which have a label *greater* than the label of the edge; this is in order to respect the intuition that $F(a) \Rightarrow F(a \times b)$, forbidding an action implies forbidding any action that is greater. The last line states that if the prohibition is violated then the reparation $\mathcal{C}$ must hold in the states where the violation is observed.

For the semantics of P we specify that $\bullet$ markers should not be present in order to capture the principle that *what is not forbidden is permitted*. The semantics of O, P, or F hints at the trace-based semantics of Process Logic [Pra79] and to some extent to the modalities of [vdM96].

We may see the semantics given in terms of "actions as trees" as a unification of the two semantics known for Dynamic Logics: the one given in terms of relations over the states of the Kripke structure, and the other given in terms of traces over the Kripke structure. The semantics for our language combines the two: for O, P, or F the semantics is given in terms of traces where for $[\cdot]$ or $\langle\cdot\rangle$ the semantics is given in terms of relations.

Satisfiability for contracts, intuitively means that given a model of a contract in the form of a normative structure then a contract (or contract clause) *can* be respected in a given world w if it is satisfied in that world. Moreover, a contract can be respected *any time* in the model (starting in any world) iff it is respected in any world of the model.

Satisfiability is a practically useful concept for contracts as informally it models that fact that for a given contract we ask if there exists some some particular behavior of the parties in the contract that can respect the contract (i.e. satisfy the contract). This guarantes that the contract is not some fictive contract which cannot be respected under any means.

Note that we are using the word "can" when we are talking about satisfiability w.r.t. normative structures because these may contain violating runs. In next section we expand on violating/accepting runs in the context of *normative structures*.

The validity problem for contracts is not so interesting. At an intuitive understanding, validity says that whatever model of a contract one takes the original contract (clause) $\mathcal{C}$ will be respected in any world of this model. Picking any normative structure $K^{\mathcal{N}}$ is the same as saying that we pick an arbitrary valuation function $\mathcal{V}$ for the propositional simbols in the worlds. This makes the valid contract a not interesting contract. Intuitively it means that this valid contract cannot be violated. Whatever parties one takes with whatever behavior, they will still respect the contract. There is no *freedom of contract* which is a concept often found in the legal comunity.

We say that a set of formulas Δ is satisfiable in a model $K^{\mathcal{N}}$ and denote it by $K^{\mathcal{N}} \models \Delta$ iff for all formulas $\mathcal{C} \in \Delta$ we have that $K^{\mathcal{N}} \models \mathcal{C}$. We say that a set of formulas Δ *entails* another formula $\mathcal{C}$, and denote it by $\Delta \models \mathcal{C}$ iff $\forall K^{\mathcal{N}}$ s.t. $K^{\mathcal{N}} \models \Delta$ then $K^{\mathcal{N}} \models \mathcal{C}$. We call the theory of Δ and denote it by $\mathbf{Th}^{\mathcal{N}}$ the set of all formulas entailed by Δ; i.e. $\mathbf{Th}^{\mathcal{N}} = \{\mathcal{C} \mid \Delta \models \mathcal{C}\}$. We denote by $\|\mathcal{C}\|^{\mathcal{N}}$ the set of all normative structures which are models of the formula $\mathcal{C}$; i.e. $\|\mathcal{C}\|^{\mathcal{N}} = \{K^{\mathcal{N}} \mid K^{\mathcal{N}} \models \mathcal{C}\}$. In contract terminology we say that a contract $\mathcal{C}'$ *conforms* with another contract $\mathcal{C}$ iff $\mathcal{C}' \models \mathcal{C}$; i.e. $\mathcal{C}'$ logicaly entails $\mathcal{C}$.

Satisfiability and validity of contracts are strong related to the tableau based proof methods. In this direction the work of Rajeev Gore on the TWB tool [AG07, AGW07] is relevant. The TWB takes as input the syntax of the logic and the tableau proof rules and returns a proof checker. Following this work it would be rather straightforward to obtain a proof checker for our $\mathcal{CL}$.

The *model-checking problem* for $\mathcal{CL}$ is interesting in practice as it says: given a model $K^{\mathcal{N}}$ and a contract $\mathcal{C}$ is it the case that the contract is satisfied by the model; $K^{\mathcal{N}} \models \mathcal{C}$? Informally this means that given a contract and given some specific parties with some particular behavior, the model-checking problem will decide if the parties will respect the contract. Moreover, the model-checking problem will give a counter example in case the answer to the question is negative. The counterexample is used to change the particular behavior of the parties such that they will respect the contract. Note that the same contract may be violated if we chnage only one of the parties. This make the contract rather specific for some particular contracting situation. This gives *freedom of contract.*

Another interesting problem related to model-checking and to the satisfiability problem is the problem of finding *all* the models which satisfy a contract. This amount to finding all the possible bahaiours of the contracting parties such that they do not violate the contract.

3.1.1 Properties of the branching semantics

We state first some properties of the $\mathcal{CL}$ language which have been proven in [Pri08a].

Proposition 3.1 ([Pri08a])**.** *The following statements are true:*

$$\models \neg O_{\mathcal{C}}(\mathbf{0}) \tag{2}$$
$$\models O_{\mathcal{C}}(\mathbf{1}) \tag{3}$$
$$\models P(\alpha) \rightarrow \neg F_{\mathcal{C}}(\alpha) \tag{4}$$
$$\models O_{\mathcal{C}}(\alpha) \rightarrow P(\alpha) \tag{5}$$
$$\textit{if } \alpha = \beta \textit{ then } \models O_{\mathcal{C}}(\alpha) \leftrightarrow O_{\mathcal{C}}(\beta) \tag{6}$$
$$\models O_{\mathcal{C}}(\alpha) \rightarrow \neg F_{\mathcal{C}}(\alpha) \tag{7}$$
$$\models O_{\mathcal{C}}(\alpha) \wedge O_{\mathcal{C}}(\beta) \rightarrow O_{\mathcal{C}}(\alpha \times \beta) \tag{8}$$
$$\models F_{\mathcal{C}}(\alpha) \rightarrow F_{\mathcal{C}}(\alpha \times \beta) \tag{9}$$
$$\models F_{\mathcal{C}}(\alpha + \beta) \leftrightarrow F_{\mathcal{C}}(\alpha) \wedge F_{\mathcal{C}}(\beta) \tag{10}$$
$$\models P(\alpha + \beta) \leftrightarrow P(\alpha) \wedge P(\beta) \tag{11}$$

The following point out conflicts that are avoided in $\mathcal{CL}$:

$$\models \neg(O_{\mathcal{C}}(\alpha) \wedge F_{\mathcal{C}}(\alpha)) \tag{12}$$
$$\models \neg(P(\alpha) \wedge F_{\mathcal{C}}(\alpha)) \tag{13}$$
$$\textit{if } \alpha \#_{\mathcal{C}} \beta \textit{ then } \models \neg(O_{\mathcal{C}}(\alpha) \wedge O_{\mathcal{C}}(\beta)) \tag{14}$$

The following result shows that the semantics avoids several *unwanted implications.*

Proposition 3.2 ([Pri08a])**.** *The following statements are true:*

$$\not\models O_{\mathcal{C}}(\alpha) \rightarrow O_{\mathcal{C}}(\alpha \times \beta) \tag{15}$$
$$\not\models O_{\mathcal{C}}(\alpha \times \beta) \rightarrow O_{\mathcal{C}}(\alpha) \tag{16}$$
$$\not\models O_{\mathcal{C}}(\alpha + \beta) \rightarrow O_{\mathcal{C}}(\alpha \times \beta) \tag{17}$$
$$\not\models O_{\mathcal{C}}(\alpha \times \beta) \rightarrow O_{\mathcal{C}}(\alpha + \beta) \tag{18}$$
$$\not\models O_{\mathcal{C}}(\alpha) \rightarrow O_{\mathcal{C}}(\alpha + \beta) \tag{19}$$
$$\not\models O_{\mathcal{C}}(\alpha + \beta) \rightarrow O_{\mathcal{C}}(\alpha) \tag{20}$$
$$\not\models F_{\mathcal{C}}(\alpha \times \beta) \rightarrow F_{\mathcal{C}}(\alpha) \tag{21}$$
$$\not\models P(\alpha \times \beta) \rightarrow P(\alpha) \tag{22}$$

With the above semantics we cannot prove validity of expressions like $F(a) \wedge (\varphi \Rightarrow P(a))$ which may be intuitive for the reader as (s)he may think of real examples where some action a is declared forbidden and only in some exceptional cases (φ holds) it is permitted. It is easy to see that there exists a model in which the formula is not satisfied. This is because the semantics of $F(a)$ requires to have the propositional constant $\mathcal{F}_a$ in a state where the semantics of $P(a)$ requires to have in the same state the negation $\neg\mathcal{F}_a$ which is impossible. In this case the same intuitive example can be modelled in $\mathcal{CL}$ as $(\neg\varphi \Rightarrow F(a)) \wedge (\varphi \Rightarrow P(a))$ which is easily proven valid from a logical point of view is also more natural.

Proposition 3.3 (Properties of obligations)**.**

$$O_{\mathcal{C}}(a+b) \not\models O_{\mathcal{C}}(a) \oplus O_{\mathcal{C}}(b) \tag{23}$$
$$O_{\mathcal{C}}(a) \oplus O_{\mathcal{C}}(b) \not\models O_{\mathcal{C}}(a+b) \tag{24}$$

With the above semantics we have the following:

Proposition 3.4.

$$F(a \cdot b) \text{ iff } F(a) \vee \langle a \rangle F(b) \tag{25}$$
$$P(a \cdot b) \text{ iff } P(a) \wedge [a]P(b) \tag{26}$$

Proof: All are easily proven by following the semantics above and the classical semantics for the propositional operators. For equation **??** for example the proof has to follow the standard way that $\forall K^{\mathcal{N}}$ a model of $F(a)$ we must prove that it is also a model of $F(a\&b)$, i.e. if $K^{\mathcal{N}} \models F(a)$ then $K^{\mathcal{N}} \models F(a\&b)$. □

Theorem 3.5 (tree model property)**.** *If a formula $\mathcal{C}$ has a model $K^{\mathcal{N}}$ then it has a tree model $\mathcal{T}^{\mathcal{N}}$.*

Theorem 3.6 (finite model theorem)**.** *If a formula $\mathcal{C}$ has a model then it has a finite model.*

3.2 Trace semantics in terms of respecting traces

The present section is devoted to presenting a semantics for $\mathcal{CL}$ with the goal of monitoring electronic contracts. For this we are interested in identifying the *respecting* and *violating* traces of actions. Obviously we do not capture everything that we do with the semantics based on normative structures of Section 3.1. We relate the two semantics in Section 3.3. We follow the many works in the literature which have a presentation based on traces e.g. [Pra79, VdM90, Eme90, Maz88]. The contents of this section has been presented in [KPS08a] and detailed in [KPS08b].

Definition 3.5 (traces)**.** *Consider a* trace *denoted $\sigma = a_0, a_1, \ldots$ as an ordered sequence of concurrent actions and skip. Formally a trace is a map $\sigma : \mathbb{N} \rightarrow \mathcal{A}_B^{\&} \cup \{\mathbf{1}\}$ from natural numbers (denoting positions) to concurrent actions from $\mathcal{A}_B^{\&}$. Take $|\sigma| \in \mathbb{N} \cup \infty$ to be the length of a trace. A (infinite) trace which from some position i onwards has only action $\mathbf{1}$ is considered* finite *and its length is i. We use ε to denote the* empty trace. *We denote by $\sigma(i)$ the* element *of a trace at position i, by $\sigma(i..j)$ a finite* subtrace, *and by $\sigma(i..)$ the infinite subtrace starting at position i in σ. The* concatenation *of two traces σ' and σ'' is denoted $\sigma'\sigma''$ and is defined iff the trace σ' is finite; $\sigma'\sigma''(i) = \sigma'(i)$ if $i < |\sigma'|$ and $\sigma'\sigma''(i) = \sigma''(i - |\sigma'|)$ for $i \geq |\sigma'|$ (e.g. $\sigma(0)$ is the first action of a trace, $\sigma = \sigma(0..i)\sigma'$ where $\sigma' = \sigma(i{+}1..)$).*

One may notice that a trace as defined above is exactly a concurrent string as in Section 3.1 where sets of concurrent strings are the models of the synchronous actions. In the context of this paper for run-time monitoring we are interested in single traces; whether one trace (concurrent string) respects (or violates) a contract clause. We see the formal definition of these notions in the sequel. We are looking only at the (traces of) actions of the parties involved in the contract and we only want to know if a trace violates (or respects) a contract.

It may be intuitive that the set of traces associated to an action α is $\|I_{\mathcal{CA}}(\alpha)\|$ (where $I_{\mathcal{CA}}$ is the interpretation of the deontic actions from Theorem 2.5 and the $\|\cdot\|$ denotes all final paths in the tree), but it is not so intuitive what is the set of traces for a negation of an action $\overline{\alpha}$. The following result shows us explicitly the set of traces which correspond to the action negation. The result helps in the proof of the next result in Proposition 3.8. The two propositions below help in understanding better the semantic definition of $[\overline{\alpha_\&}]\mathcal{C}$.

Proposition 3.7 (action negation as traces)**.** *The set* $\|I_{\mathcal{CA}}(\overline{\alpha})\|$ *of traces which are full paths of the tree interpreting the negation of* $\underline{\alpha} = +_{i\in I}\ \alpha^i_\& \cdot \alpha^i$ *is equal to the following set of traces:*

$$\begin{aligned}\{\overline{\alpha}\} =& \{\sigma \mid \sigma = \sigma(0)\varepsilon\ \wedge\ \forall i \in I,\ \alpha^i_\& \not\subseteq \sigma(0)\}\ \cup\\ &\{\sigma \mid \sigma = \sigma(0)\sigma(1..)\ \wedge\ \exists i \in I,\ s.t.\ \alpha^i \neq \mathbf{1} \wedge \alpha^i_\& \subseteq \sigma(0)\\ &\qquad \wedge\ \sigma(1..) \in \{\overline{+_{i\in I'}\ \underline{\alpha^i}}\},\ I' = \{i \in I \mid \alpha^i_\times \subseteq \sigma(0)\}\}.\end{aligned}$$

Proof: The definition of the set of traces is inductive. This is because the definition of the canonical form of actions is inductive and therefore also the action negation. The tree $I_{\mathcal{CA}}(\overline{\alpha})$ is the same as $I_{\mathcal{CA}}(\overline{+_{i\in I}\ \alpha^i_\& \cdot \alpha^i})$ which is $I_{\mathcal{CA}}(+_{\gamma\in\overline{R}}\gamma\ +\ +_{j\in J}\ \gamma^j_\times \cdot \overline{+_{i\in I'}\ \underline{\alpha^i}})$. We give an intuitive analogy between the set of traces defined here and the definition of action negation from Definition 2.5. The first part of the action negation, i.e. $+_{\gamma\in\overline{R}}\gamma$ gives the full paths of the tree of length 1 (i.e. on the first level). It is simple to observe that these paths are captured by the first set of traces $\{\sigma \mid \sigma = \sigma(0)\varepsilon\ \wedge\ \forall i \in I,\ \alpha^i_\& \not\subseteq \sigma(0)\}$. These are traces with one element $\sigma(0)$ (i.e. ending in the empty trace ε) and they respect the same condition like in the definition of $\overline{R}$. Note that when I is a singleton and we have only one $\alpha_\&$ then the condition $\alpha_\& \not\subseteq \sigma(0)$ becomes $\sigma(0) = \sigma'(0) \cup \sigma''(0) \wedge \sigma'(0) \subset \alpha_\& \wedge \sigma''(0) \subseteq \mathcal{A}^\&_B \backslash \alpha_\&$. We read this condition as: the first element of σ (which we recall is a set of basic actions) has some actions, but not all, i.e. $\sigma'(0)$, among those of the basic actions of $\alpha_\&$ and the other basic actions, i.e. $\sigma''(0)$ are different than those of $\alpha_\&$; i.e. are among $\mathcal{A}^\&_B \backslash \alpha_\&$.

The other full paths of length greater than 1 of the tree are given by the second part of the action, i.e. $+_{j\in J}\ \gamma^j_\times \cdot \overline{+_{i\in I'}\ \underline{\alpha^i}}$. All these paths are captured by the second set of traces which are of length at least 2. All branches of $+_{i\in I}\, \alpha^i_\& \cdot \overline{\alpha^i}$ where $\alpha^i = \mathbf{1}$ do not contribute to the set of traces. This is because the negation $\overline{\alpha^i + \ldots} = \overline{\mathbf{1} + \ldots} = \mathbf{0}$ meaning that the branch ends in $\mathbf{0}$ which propagates upwards by $\alpha \cdot \mathbf{0} = \mathbf{0}$ and dissapears eventually by $\alpha + \mathbf{0} = \alpha$. Therefore we are looking only at traces s.t. $\alpha^i \neq \mathbf{1}$. From these we take the traces which start with the action that includes $\alpha^i_\&$ (i.e. $\alpha^i_\& \subseteq \sigma(0)$) and are followed by a trace (i.e. $\sigma(1..)$) which is part of the traces of the negation of the smaller compound action $+_{i\in I'}\ \underline{\alpha^i}$. The α^i are related to $\sigma(0)$ by the indexing set $I' = \{i \in I \mid \alpha^i_\times \subseteq \sigma(0)\}$. □

The next result of Proposition 3.8 is useful in giving us the completeness of the semantical definition for $[\alpha_\&]\mathcal{C}$. Particularly, we get as an immediate corollary that the conditions on traces from the semantic definition of $[\alpha_\&]\mathcal{C}$ cover all the possible traces. The same result is used for the semantical definitions of $[\overline{\alpha_\&}]\mathcal{C}$, $O_\mathcal{C}(\alpha_\&)$, or $F_\mathcal{C}(\alpha_\&)$. Particularly, it explains why the two conditions in the semantical definition of $O_\mathcal{C}(\alpha_\&)$ are complementary and therefore why the second condition does not need the explicit specification of the negation of the

$\sigma \models \top$	
$\sigma \not\models \bot$	
$\sigma \models \mathcal{C}_1 \rightarrow \mathcal{C}_2$	if whenever $\sigma \models \mathcal{C}_1$ then $\sigma \models \mathcal{C}_2$.
$\sigma \models \mathcal{C}_1 \wedge \mathcal{C}_2$	if $\sigma \models \mathcal{C}_1$ and $\sigma \models \mathcal{C}_2$.
$\sigma \models \mathcal{C}_1 \vee \mathcal{C}_2$	if $\sigma \models \mathcal{C}_1$ or $\sigma \models \mathcal{C}_2$.
$\sigma \models \mathcal{C}_1 \oplus \mathcal{C}_2$	if ($\sigma \models \mathcal{C}_1$ and $\sigma \not\models \mathcal{C}_2$) or ($\sigma \not\models \mathcal{C}_1$ and $\sigma \models \mathcal{C}_2$).
$\sigma \models [\alpha_\&]\mathcal{C}$	if $\alpha_\& = \sigma(0)$ and $\sigma(1..) \models \mathcal{C},$ or $\alpha_\& \neq \sigma(0)$.
$\sigma \models [\beta \cdot \beta']\mathcal{C}$	if $\sigma \models [\beta][\beta']\mathcal{C}$.
$\sigma \models [\beta + \beta']\mathcal{C}$	if $\sigma \models [\beta]\mathcal{C}$ and $\sigma \models [\beta']\mathcal{C}$.
$\sigma \models [\beta^*]\mathcal{C}$	if $\sigma \models \mathcal{C}$ and $\sigma \models [\beta][\beta^*]\mathcal{C}$.
$\sigma \models [\mathcal{C}_1?]\mathcal{C}_2$	if $\sigma \not\models \mathcal{C}_1,$ or $if \sigma \models \mathcal{C}_1$ and $\sigma \models \mathcal{C}_2$.
$\sigma \models O_{\mathcal{C}}(\alpha_\&)$	if $\alpha_\& \subseteq \sigma(0),$ or $if \sigma(1..) \models \mathcal{C}$.
$\sigma \models O_{\mathcal{C}}(\alpha \cdot \alpha')$	if $\sigma \models O_{\mathcal{C}}(\alpha)$ and $\sigma \models [[\alpha]]O_{\mathcal{C}}(\alpha')$.
$\sigma \models O_{\mathcal{C}}(\alpha + \alpha')$	if $\sigma \models O_\bot(\alpha)$ or $\sigma \models O_\bot(\alpha')$ or $\sigma \models [\overline{\alpha + \alpha'}]\mathcal{C}$.
$\sigma \models F_{\mathcal{C}}(\alpha_\&)$	if $\alpha_\& \not\subseteq \sigma(0),$ or $if \alpha_\& \subseteq \sigma(0)$ and $\sigma(1..) \models \mathcal{C}$.
$\sigma \models F_{\mathcal{C}}(\alpha \cdot \alpha')$	if $\sigma \models F_\bot(\alpha)$ or $\sigma \models [[\alpha]]F_{\mathcal{C}}(\alpha')$.
$\sigma \models F_{\mathcal{C}}(\alpha + \alpha')$	if $\sigma \models F_{\mathcal{C}}(\alpha)$ and $\sigma \models F_{\mathcal{C}}(\alpha')$.

Table 4: Trace semantics of $\mathcal{CL}$.

concurrent action. For convenience we define an *enclosing* relation over traces $\sigma \supseteq \sigma'$ iff $\forall i \in \mathbb{N}, \sigma'(i) \subseteq \sigma(i)$. Note that this definition requires that $m_\sigma \geq m_{\sigma'}$.

Proposition 3.8. *For an arbitrary action α, any infinite trace σ is either starting with a trace bigger w.r.t. $\supseteq$ then a complete path of $I_{\mathcal{CA}}(\alpha)$ or it starts with a trace bigger than a complete path of $I_{\mathcal{CA}}(\overline{\alpha})$.*

Definition 3.6 (Semantics of $\mathcal{CL}$)**.** *We give in Table 4 a recursive definition of the* satisfaction relation $\models$ *over pairs $(\sigma, \mathcal{C})$ of a trace and a contract; it is usually written $\sigma \models \mathcal{C}$ and it is read as "trace σ* respects *the contract (clause) $\mathcal{C}$". We write $\sigma \not\models \mathcal{C}$ instead of $(\sigma, \mathcal{C}) \notin \models$ and read it as "σ* violates *$\mathcal{C}$."*

The semantics definition for the propositional operators is classical. Recal the definition of the negation operator $\neg\mathcal{C} \triangleq \mathcal{C} \rightarrow \bot$. Now it is simple to see the semantics of negation as:

$$\sigma \models \neg\mathcal{C} \quad \text{if} \quad \sigma \not\models \mathcal{C}$$

The semantics of the dynamic logic operators is not so common. Standard PDL is a branching time logic interpreted over labeled Kripde structures. Standard interpretation of the $[\beta]\mathcal{C}$ is that it holds in the current state if *all* states reachable by β then $\mathcal{C}$ holds. The dual $\langle\beta\rangle\mathcal{C} = \neg[\beta]\neg\mathcal{C}$ has the interpretation true in the current state iff *exists* a state reachable by β where $\mathcal{C}$ holds. Here we interpret the two dynamic modalities over linear structures, i.e. over traces (not sets of traces as in [Pra79] which is the same as interpreting in a Kripke structure). The smallest syntactic element that needs semantics is the modality over concurrent actions $[\alpha_\times]$ which also includes the basic actions. A trace σ respects the formula $[\alpha_\times]\mathcal{C}$ if either the first (set of actions) element of the trace $\sigma(0)$ is not equal with the (set of basic actions forming the) action $\alpha_\times$ or otherwise $\sigma(0)$ is the same as action $\alpha_\times$ and $\mathcal{C}$ is respected by the rest of the trace (i.e. $\sigma(1..) \models \mathcal{C}$). The semantics of the compound actions is given compositionaly in the classical way found in PDL. The semantics of the dual operator $\langle\alpha_\times\rangle\mathcal{C}$ is:

$$\sigma \models \langle\alpha_\times\rangle\mathcal{C} \text{ iff } \sigma \models \neg[\alpha_\times]\neg\mathcal{C} \text{ iff } \alpha_\times = \sigma(0) \text{ and } \sigma(1..) \models \mathcal{C}$$

For the compound actions under $\langle\cdot\rangle$ just dualize the corresponding semantic definitions for $[\cdot]$.

Denote by $\mathbf{any} = +_{\alpha_\times \in \mathcal{A}_B^\times} \alpha_\times$ the choice between *all* the concurrent actions. For all $\alpha_\times \in \mathcal{A}_B^\times$ denote by $\langle\langle\alpha_\times\rangle\rangle\phi$ the formula $\langle\alpha_\times \times \mathbf{any}\rangle\phi$ and by $[[\alpha_\times]]\phi$ the formula $[\alpha_\times \times \mathbf{any}]\phi$. Note that $\langle\langle\cdot\rangle\rangle$ and $[[\cdot]]$ are duals in this definition. The extension of $[[\cdot]]$ to all actions $\alpha \in \mathcal{A}$ is done in the classical way of PDL as was done for $[\cdot]$ (a similar definition is given for $\langle\langle\cdot\rangle\rangle$):

$$\begin{aligned}
[[\alpha + \beta]]\phi &= [[\alpha]]\phi \wedge [[\beta]]\phi \\
[[\alpha \cdot \beta]]\phi &= [[\alpha]][[\beta]]\phi \\
[[\alpha^*]]\phi &= \phi \wedge [[\alpha]][[\alpha^*]]\phi
\end{aligned}$$

A trace σ respects an obligation $O_\mathcal{C}(\alpha_\&)$ if either of the two complementary conditions (see Proposition 3.8) is satisfied. The first condition deals with the obligation itself: the trace σ respects the obligation $O(\alpha_\&)$ if the first action of the trace includes $\alpha_\&$. Otherwise, in case the obligation is violated,[5] the only way to fulfill the contract is by respecting the reparation $\mathcal{C}$; i.e. $\sigma(1..) \models \mathcal{C}$.

An important requirement when modelling electronic contracts is that the obligation of a sequence of actions $O_\mathcal{C}(\alpha \cdot \alpha')$ must be equal to the obligation of the first action $O_\mathcal{C}(\alpha)$ and after the first obligation is respected the second obligation must hold $O_\mathcal{C}(\alpha')$. To respect the obligation $O_\mathcal{C}(\alpha)$ means to do any action bigger than α which is captured with the syntactic construction $[[\cdot]]$. Note that if $O_\mathcal{C}(\alpha)$ is violated then it is required that the second obligation is discarded, and the reparation $\mathcal{C}$ must hold. Violating $O_\mathcal{C}(\alpha)$ means that α is not executed and thus, by the semantic definition, $[[\overline{\alpha}]]O_\mathcal{C}(\alpha')$ holds regardless of $O_\mathcal{C}(\alpha')$.

Respecting an obligation of a choice action $O_\mathcal{C}(\alpha_1 + \alpha_2)$ means that it must be executed one of the actions α_1 or α_2 completely; i.e. obligation needs to consider only one of the choices. We use the $\perp$ as a reparation for the obligations of the smaller actions $O_\perp(\alpha)$ which means that the actions should be executed *completely* (otherwise the obligation is violated). If none of these is entirely executed then a violation occurs (thus the negation of the action is needed) so the reparation $\mathcal{C}$ must be respected. We use the operator $[\cdot]$ over the action negation which looks at *exactly* the actions specified by $\overline{\alpha + \alpha'}$. We do not need to look at bigger actions (as we do for $O_\mathcal{C}(\alpha_\times)$ or for $O_\mathcal{C}(\alpha \cdot \alpha')$ with $[[\cdot]]$) because the action negation encodes all these (i.e. the second part of the Definition 2.5).

The semantics is particularly defined such that it captures some intuitive properties one finds in legal contracts; we list them in Proposition 3.9. We say

[5] Violation of an obligatory action is encoded by the action negation.

that a formula $\mathcal{C}$ is *valid* and denote it by $\models \mathcal{C}$ iff $\forall\sigma,\ \sigma \models \mathcal{C}$. A formula is not valid, denoted $\not\models \mathcal{C}$ iff $\exists\sigma$ s.t. $\sigma \not\models \mathcal{C}$.

Proposition 3.9 (properties on traces)**.**

$$\models O_{\mathcal{C}}(a) \wedge O_{\mathcal{C}}(b) \Leftrightarrow O_{\mathcal{C}}(a\&b) \tag{27}$$
$$\models F(a) \Rightarrow F(a\&b) \tag{28}$$
$$\models F(a+b) \Leftrightarrow F(a) \wedge F(b) \tag{29}$$
$$\models F(a \cdot b) \Leftrightarrow F(a) \vee [a]F(b) \tag{30}$$
$$\models [[\alpha_{\&}]]\mathcal{C} \Rightarrow [[\alpha_{\&}\&\alpha'_{\&}]]\mathcal{C} \tag{31}$$
$$\not\models O(a+b) \Rightarrow O(a\&b) \tag{32}$$
$$\not\models O(a+b) \Rightarrow O(a) \tag{33}$$
$$\not\models F(a\&b) \Rightarrow F(a) \tag{34}$$

Proof: The proofs of these properties is routine. The method is the classical one for validity of implication where we need to look at all and only the models which satisfy the formula on the left of the implication and make sure that they satisfy also the formula on the right.

For property 27: We must prove two implications. We deal first with the $\Rightarrow$ one. Take a trajectory σ s.t. it satisfies the formula on the left, i.e. $\sigma \models O_{\mathcal{C}}(a)$ and $\sigma \models O_{\mathcal{C}}(b)$. We are in the simple case when we consider basic actions a and b. We look at the semantics of obligation. If it is the case that $\sigma(1..) \models \mathcal{C}$ than it is clear that $\sigma \models O(a\&b)$. Otherwise we have the case when both $\sigma(0) \supseteq a$ and $\sigma(0) \supseteq b$. It implies that $\sigma(0) \supseteq a\&b$ which means that $\sigma \models O(a\&b)$. The second implication $\Leftarrow$ is simpler and uses the same judgement.

Properties 28, 29, or 30 are similar.

For property 32 we need to give a counterexample. Clearly a trace starting with $\sigma(0) = a$ satisfies $O(a+b)$ but does not satisfy $O(a\&b)$. We find similar counterexamples for properties like 33 or 34. For property 31 we again need to show that for all σ which respect the formula on the left of the arrow they also respect the formula on the right. $\square$

From [HKT00] we know how to encode LTL only with the dynamic $[\cdot]$ modality and the Kleene *; e.g. "*always* obliged to do α" is encoded as $[\mathbf{any}^*]O(\alpha)$.

Example 1. as traces: Consider the expression $[e]O_{O_{\perp}(p\cdot p)}(p + d\&n)$ which encodes in $\mathcal{CL}$ the contract clause of Example 1. from the introduction. We give here few examples of traces of actions which *respect* the contract clause:

- e, p ("exceed bandwidth limit" and then "pay") which respects the contract because it respects the top level obligation;
- e, d, p, p ("exceed bandwidth limit" and then "delay payment" after which "pay" twice in a row) which even if it violates the top level obligation because it does not notify by e-mail at the same time when "delaying payment", it still respects the reparation by paying twice;
- p, p, p ("pay" three times in a row) because every trace which does not start with the action e respects the contract.

Examples of traces which *violate* the clause are:

- e, e, e (constantly "exceeding the bandwidth limit") which violates both the first obligation and the second one by not paying;

- e, d, d (after "exceeding the bandwidth limit" it constantly "delays the payment") which again violates both obligations.

3.3 Relating the trace and the branching semantics

Theorem 3.10 (Relating the linear and the branching semantics)**.**
If $\sigma \models \mathcal{C}$ then $\forall K^{\mathcal{N}}$ and $\forall w \in K^{\mathcal{N}}$ s.t. $K^{\mathcal{N}}, w \models \mathcal{C}$ then $\sigma \in K^{\mathcal{N}}$.

Corollary 3.11. *For any formula $\mathcal{C}$ we have that*

$$\bigcap_{\mathcal{T}^{\mathcal{N}} \in \|\mathcal{C}\|^{\mathcal{N}}} \mathcal{T}^{\mathcal{N}} = \|\mathcal{C}\|^{\sigma}.$$

The corollary intuitively states that the intersection of all the sets of trajectories denoted by the tree models of the formula $\mathcal{C}$ is equal to the set of trajectories given by the linear semantics.

4 Conclusion

By now we have presented the $\mathcal{CL}$ action-based logic which can be used for reasoning about contracts. Case study examples of how one can use $\mathcal{CL}$ for writing specifications of legal contracts are done elsewhere (e.g. see [PS07c, PPS07]). The theoretical discourse was concerned only with defining the semantics of the $\mathcal{CL}$ logics. We have give two different semantics (a branching and a linear semantics) with completely different purposes (respectively run-time monitoring of contracts and static reasoning about contracts). The semantics are related in the end of the report which gives confidence in the correctness of their definitions.

Further work is directed towards building theoretical tool on top of the two semantics. A first step is in building a tableau proof system for the branching semantics.

4.1 Related Work

Some of the most known and studied action algebras come from the work on dynamic logics [Pra76]. We base our work on Kleene algebra which was introduced by Kleene in 1956 and further developed by Conway in [Con71]. For references and an introduction to Kleene algebra see the extensive work of Kozen [Koz80, Koz90, Koz97]. In these research efforts the authors used, for example, regular languages as the objects of the algebra, or relations over a fixed set and analyze properties like completeness [Koz94], complexity [CKS96] and applications [Coh94] of variants of Kleene algebra. Some variants include the notion of *tests* [Koz97], and others add some form of types or discard the identity element **1** [Koz98]. An interpretation for Kleene algebra with tests has been given using automata over guarded strings [Koz03]. An introduction to the method of giving interpretation using trees and operations on trees can be found in [Hen88].

Acknowledgements

We would like to thank Martin Steffen for the very fruitful discussions we had and to Sergiu Bursuc for the many observations on earlier drafts of this work. We would also like to thank the reviewers for pointing out other interesting and relevant related work and for helping shape the report and the presentation as it is now.

References

[AG07] Pietro Abate and Rajeev Gore. Tableau work bench: Theory and practice, the. In Nicola Olivetti, editor, *International Conference TABLEAUX 2007*, Lecture Notes in Artificial Intelligence. Springer, 2007.

[AGW07] Pietro Abate, Rajeev Gore, and Florian Widmann. Single-pass tableaux for computation tree logic. In Nachum Dershowitz and Andrei Voronkov, editors, *14th International Conference on Logic for Programming Artificial Intelligence and Reasoning*, Lecture Notes in Computer Science. Springer, 2007.

[BAHP81] Mordechai Ben-Ari, Joseph Y. Halpern, and Amir Pnueli. Finite models for deterministic propositional dynamic logic. In Shimon Even and Oded Kariv, editors, *8th Colloquium On Automata, Languages and Programming (ICALP'81)*, volume 115 of *Lecture Notes in Computer Science*, pages 249–263. Springer, 1981.

[Ber00] Gérard Berry. The foundations of Esterel. In *Proof, language, and interaction: essays in honour of Robin Milner*, pages 425–454. MIT Press, 2000.

[BN98] Frantz Baader and Tobias Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[Bro03] Jan Broersen. *Modal Action Logics for Reasoning About Reactive Systems*. PhD thesis, Vrije Universiteit Amsterdam, 2003.

[BWM01] Jan Broersen, Roel Wieringa, and John-Jules Ch. Meyer. A fixed-point characterization of a deontic logic of regular action. *Fundam. Inf.*, 48(2-3):107–128, 2001.

[CJ02] Jose Carmo and Andrew Jones. Deontic logic and contrary-to-duties. In Dov Gabbay and Franz Guenthner, editors, *Handbook of Philosophical Logic*, pages 265–343. Kluwer Academic Publishers, 2002.

[CKS81] Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.

[CKS96] Ernie Cohen, Dexter Kozen, and Frederick Smith. The Complexity of Kleene Algebra with Tests. Technical report, Cornell University, 1996.

[CM07] Pablo F. Castro and T.S.E. Maibaum. A complete and compact propositional deontic logic. In Cliff B. Jones, Zhiming Liu, and Jim Woodcock, editors, *4th International Colloquium on Theoretical Aspects of Computing (ICTAC'07)*, volume 4711 of *Lecture Notes in Computer Science*, pages 109–123. Springer-Verlag, 2007.

[Coh94] Ernie Cohen. Using kleene algebra to reason about concurrency control. Technical report, Telcordia, 1994.

[Con71] John Horton Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, 1971.

[CP01] José Carmo and Olga Pacheco. Deontic and action logics for organized collective agency, modeled through institutionalized agents and roles. *Fundam. Inf.*, 48(2-3):129–163, 2001.

[dMW96] P. d'Altan, John-Jules Ch. Meyer, and Roel Wieringa. An integrated framework for ought-to-be and ought-to-do constraints. *Artif. Intell. Law*, 4(2):77–111, 1996.

[Eme90] E. Allen Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 995–1072. Elsevier, 1990.

[Har83] David Harel. Recurring dominoes: Making the highly undecidable highly understandable (preliminary report). In Marek Karpinski, editor, *Fundamentals of Computation Theory (FCT'83)*, volume 158 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 1983.

[Hen88] Matthew Hennessy. *Algebraic Theory of Processes.* MIT Press, 1988.

[HKT00] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic.* MIT Press, 2000.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[Koz80] Dexter Kozen. A representation theorem for models of *-free PDL. In J. W. de Bakker and Jan van Leeuwen, editors, *7th Colloquium on Automata, Languages and Programming (ICALP'80)*, volume 85 of *Lecture Notes in Computer Science*, pages 351–362. Springer, 1980.

[Koz83] Dexter Kozen. Results on the propositional mu-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983.

[Koz90] Dexter Kozen. On kleene algebras and closed semirings. In Branislav Rovan, editor, *Mathematical Foundations of Computer Science (MFCS'90)*, volume 452 of *Lecture Notes in Computer Science*, pages 26–47. Springer, 1990.

[Koz94] Dexter Kozen. A completeness theorem for kleene algebras and the algebra of regular events. *Information and Computation*, 110(2):366–390, 1994.

[Koz97] Dexter Kozen. Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems (TOPLAS'97)*, 19(3):427–443, 1997.

[Koz98] Dexter Kozen. Typed kleene algebra. Technical Report 1669, Computer Science Department, Cornell University, March 1998.

[Koz03] Dexter Kozen. Automata on Guarded Strings and Applications. In John T. Baldwin, Ruy J. G. B. de Queiroz, and Edward H. Haeusler, editors, *Workshop on Logic, Language, Informations and Computation (WoLLIC'01)*, volume 24 of *Matematica Contemporanea.* Sociedade Brasileira de Matemática, 2003.

[KPS08a] Marcel Kyas, Cristian Prisacariu, and Gerardo Schneider. Run-time monitoring of electronic contracts. In *ATVA'08*, volume 5311 of *LNCS*, pages 397–407. Springer-Verlag, 2008.

[KPS08b] Marcel Kyas, Cristian Prisacariu, and Gerardo Schneider. Run-time monitoring of electronic contracts - theoretical results. Technical Report 378, Department of Informatics, University of Oslo, Oslo, Norway, November 2008.

[LW04] Carsten Lutz and Dirk Walther. PDL with negation of atomic programs. In David A. Basin and Michaël Rusinowitch, editors, *2nd International Joint Conference on Automated Reasoning (IJCAR'04)*, volume 3097 of *Lecture Notes in Computer Science*, pages 259–273. Springer, 2004.

[Maz88] Antoni W. Mazurkiewicz. Basic notions of trace theory. In J. W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg, editors, *REX Workshop*, volume 354 of *Lecture Notes in Computer Science*, pages 285–363. Springer, 1988.

[Mey88] John-Jules Ch. Meyer. A different approach to deontic logic: Deontic logic viewed as a variant of dynamic logic. *Notre Dame Journal of Formal Logic*, 29(1):109–136, 1988.

[Mil83] Robin Milner. Calculi for synchrony and asynchrony. *Theorethical Computer Science*, 25:267–310, 1983.

[Mil95] Robin Milner. *Communication and concurrency.* Prentice Hall, 1995.

[OSS07] Olaf Owe, Gerardo Schneider, and Martin Steffen. Components, objects, and contracts. In *6th Workshop on Specification And Verification of Component-Based Systems (SAVCBS'07)*, ACM Digital Library, pages 91–94, Dubrovnik, Croatia, September 2007.

[Pel85] David Peleg. Concurrent dynamic logic (extended abstract). In *7th ACM Symposium on Theory of Computing (STOC'85)*, pages 232–239. ACM, 1985.

[Pel87] David Peleg. Concurrent dynamic logic. *Journal of ACM*, 34(2):450–479, 1987.

[Pnu77] Amir Pnueli. Temporal logic of programs, the. In *Proceedings of the 18th IEEE Symposium On the Foundations of Computer Science (FOCS'77)*, pages 46–57. IEEE Computer Society Press, 1977.

[PPS07] Gordon Pace, Cristian Prisacariu, and Gerardo Schneider. Model checking contracts - a case study. In Kedar Namjoshi and Tomohiro Yoneda, editors, *5th International Symposium on Automated Technology for Verification and Analysis (ATVA'07)*, volume 4762 of *Lecture Notes in Computer Science*, pages 82–97. Springer, October 2007.

[Pra76] Vaughan R. Pratt. Semantical considerations on floyd-hoare logic. In *IEEE Symposium On Foundations of Computer Science (FOCS'76)*, pages 109–121, 1976.

[Pra79] Vaughan R. Pratt. Process logic. In *6th Symposium on Principles of Programming Languages (POPL'79)*, pages 93–100. ACM, 1979.

[Pri08a] Cristian Prisacariu. Deontic modalities over synchronous actions - technicalities. Technical Report 381, Univ. Oslo, Norway, 2008.

[Pri08b] Cristian Prisacariu. Extending kleene algebra with synchrony – technicalities. Technical Report 376, Univ. Oslo, Oslo, Norway, 2008.

[PS97] Henry Prakken and Marek Sergot. Dyadic deontic logic and contrary-to-duty obligation. In Donald Nute, editor, *Defeasible Deontic Logic*, pages 223–262. Kluwer Academic Publishers, 1997.

[PS07a] Cristian Prisacariu and Gerardo Schneider. A formal language for electronic contracts. In *FMOODS'07*, volume 4468 of *LNCS*, pages 174–189. Springer, 2007.

[PS07b] Cristian Prisacariu and Gerardo Schneider. Towards a formal definition of electronic contracts. Technical Report 348, Department of Informatics, University of Oslo, Oslo, Norway, January 2007.

[Seg82] Krister Segerberg. A deontic logic of action. *Studia Logica*, 41(2):269–282, 1982.

[Seg92] Krister Segerberg. Getting started: Beginnings in the logic of action. *Studia Logica*, 51(3/4):347–378, 1992.

[VdM90] Ron Van der Meyden. Dynamic logic of permission, the. In John Mitchell, editor, *5th Annual IEEE Symp. on Logic in Computer Science, (LICS'90)*, pages 72–78. IEEE Computer Society Press, 1990.

[vdM96] Ron van der Meyden. The dynamic logic of permission. *Journal of Logic and Computation*, 6(3):465–479, 1996.

[vdT03] Leendert van der Torre. Contextual deontic logic: Normative agents, violations and independence. *Ann. Math. Artif. Intell.*, 37(1-2):33–63, 2003.

[VW68] Georg Henrik Von Wright. *An Essay in Deontic Logic and the General Theory of Action.* North Holland Publishing Co., Amsterdam, 1968.

[Wri51] Georg Henrik Von Wright. Deontic logic. *Mind*, 60:1–15, 1951.

[Wyn06] Adam Zachary Wyner. Sequences, obligations, and the contrary-to-duty paradox. In Lou Goble and John-Jules Ch. Meyer, editors, *8th International Workshop on Deontic Logic in Computer Science (DEON'06)*, volume 4048 of *Lecture Notes in Computer Science*, pages 255–271. Springer, 2006.