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

Contents

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. *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).

In [PS07b] we have introduced $\mathcal{CL}$, a formal language for writing contracts, which allows to write (conditional) obligations, permissions and prohibitions of the different contract signatories, based on the so-called *ought-to-do* approach. The *ought-to-do* approach considers the above normative notions specified over (names of) *actions*, as for example "The client is obliged to pay after each delivery". In $\mathcal{CL}$ the above would be written as $[d]O(p)$, where d is an action representing "delivery", after which $O(p)$ is the obligation of paying. Actions may be more complex, involving concurrent composition, non-deterministic choice, negation ($\overline{a}$, meaning any action but a), etc. We have also given a formal semantics of the contract language in a variant of μ-calculus.

Much of the motivations for the contract language $\mathcal{CL}$ and relevant related work can be found in [PS07b] and even more discussions and results (like deontical paradoxes we avoid) are in [PS07c]. In this report we do not give such in depth motivations and we give intuitions only to the new constructs or to notions that have changed from previous versions of $\mathcal{CL}$.

A thorough investigation of the actions underlying $\mathcal{CL}$ has been done in [PS07a] where a new algebraic structure $\mathcal{CA}$ was introduced to provide a well-founded formal basis for the action-based contract language $\mathcal{CL}$. In this report we use a slightly simpler version of the algebra $\mathcal{CA}$. We restate previous results without the proofs and give new results relevant for the rest of the report. The algebra is a crucial part of the branching semantics of $\mathcal{CL}$ from Section 3.1 and is also used in the presentation of the linear semantics of Section 3.2.

The report is organized as follows. The rest of the introduction presents briefly the $\mathcal{CL}$ contract language and an intuitive understanding of the syn-

tactical constructs.

In Section 2 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 $\mathcal{CL}$.

In the last section we conclude our work and give an extensive discussion on related works. Examples from real contracts given in the end of the introduction acompany the reader throughout the report in order to make clear most of the notions we introduce. In the end of the report we analyze in full one on the examples.

1.1 $\mathcal{CL}$ – A Formal Language for Contracts

In this section we recall the contract language $\mathcal{CL}$ first presented in [PS07b]. Here we give a slightly different and more general version, and we discuss the differences.

Definition 1.1 (Contract Language Syntax)**.** *A contract is defined by the grammar in Table 1.*

In what follows we provide an intuitive explanation of the $\mathcal{CL}$ syntax. A formal semantics is given later in Section 3.

A contract consists of two parts: *definitions* ($\mathcal{D}$) and *clauses* ($\mathcal{C}$). We deliberately let the definitions part underspecified in the syntax above. $\mathcal{D}$ specifies the *assertions* (or conditions) and the atomic actions present in the clauses. ϕ denotes assertions and ranges over boolean expressions including the usual boolean connectives, and arithmetic comparisons like "the budget is more than 200$". We let the atomic actions underspecified, which for our

$$
\begin{array}{rcl}
\mathcal{C}ontract & := & \mathcal{D} \,;\, \mathcal{C} \\
\mathcal{C} & := & \mathcal{C}_O \mid \mathcal{C}_P \mid \mathcal{C}_F \mid \mathcal{C} \wedge \mathcal{C} \mid [\beta]\mathcal{C}' \mid \langle\beta\rangle\mathcal{C}' \mid \top \mid \ \bot \\
\mathcal{C}' & := & \phi \mid \mathcal{C} \\
\mathcal{C}_O & := & O_{\mathcal{C}}(\alpha) \mid \mathcal{C}_O \oplus \mathcal{C}_O \\
\mathcal{C}_P & := & P(\alpha) \mid \mathcal{C}_P \oplus \mathcal{C}_P \\
\mathcal{C}_F & := & F_{\mathcal{C}}(\alpha) \mid \mathcal{C}_F \vee [\alpha]\mathcal{C}_F
\end{array}
$$

Table 1: Syntax grammar for the $\mathcal{CL}$ language to use in specifying contracts.

$$
\begin{array}{rcl}
\mathcal{C}ontract & := & \mathcal{D} \,;\, \mathcal{C} \\
\mathcal{C} & := & \phi \mid \mathcal{C}_O \mid \mathcal{C}_P \mid \mathcal{C}_F \mid \mathcal{C} \wedge \mathcal{C} \mid [\beta]\mathcal{C} \mid \langle\beta\rangle\mathcal{C} \mid \top \mid \ \bot \\
\mathcal{C}_O & := & O_{\mathcal{C}}(\alpha) \mid \mathcal{C}_O \oplus \mathcal{C}_O \\
\mathcal{C}_P & := & P(\alpha) \mid \mathcal{C}_P \oplus \mathcal{C}_P \\
\mathcal{C}_F & := & F_{\mathcal{C}}(\alpha) \mid \mathcal{C}_F \vee [\alpha]\mathcal{C}_F
\end{array}
$$

Table 2: Syntax grammar for the $\mathcal{CL}$ language to use in specifying properties of contracts.

purposes can be understood as consisting of three parts: the proper action, the subject performing the action, and the target of (or, the object receiving) such an action. Note that, in this way, the parties involved in a contract are encoded in the actions.

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 2 is for specifying properties of contracts. Note that an asserion ϕ alone may be a contract clause; where in the definition above this is not allowed. 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 original contracts) to have an assertion as a stand alone contract clause. In a contract, assertions are used in a restricted fashon and thus we need a second grammar for $\mathcal{CL}$ which would restrict the use of assertions ϕ. The semantics of the operators remains the same, only the allowed $\mathcal{CL}$ formulas are different.

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). This corresponds to the $\mathcal{C}'$ in Table 1.

$\mathcal{C}$ is the general *contract clause*. $\mathcal{C}_O$, $\mathcal{C}_P$, and $\mathcal{C}_F$ denote respectively *obligation*, *permission*, and *prohibition* clauses. $O_{\mathcal{C}}(\alpha)$, $P(\alpha)$, and $F_{\mathcal{C}}(\alpha)$, represents the obligation, permission, or prohibition of performing a given action

α. Intuitively $O_{\mathcal{C}}(\alpha)$ states the obligation to execute α, and the reparation $\mathcal{C}$ in case the obligation is violated, i.e. whenever α is not performed (a CTD). The reparation may be any contract clause.[1] Obligations without CTDs are 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 (like $O(\alpha)$) are (sometimes in the literature) called *categorical* because they cannot be violated (i.e. there is no reparation for their violation, thus a violation would give violation of the whole contract). Similarly, CTP statements $F_{\mathcal{C}}(\alpha)$ may be thought as the actual forbearing of the action $F(\alpha)$ together with the penalty $\mathcal{C}$ in case the prohibition is violated. The normal prohibition is a special case of CTP.

One may be tempted to define $O_{\mathcal{C}}(\alpha)$ as syntactic sugar for $O(\alpha) \wedge [\overline{\alpha}]\mathcal{C}$ which is read as "there exists the obligation of doing α and if the action α is not executed (i.e. some contradictory action is executed instead) then the reparation $\mathcal{C}$ must be enforced". Similarly, CTP statements $F_{\mathcal{C}}(\alpha)$ may be thought as $F_{\mathcal{C}}(\alpha) = F(\alpha) \wedge [\alpha]\mathcal{C}$, where $\mathcal{C}$ is the penalty in case the prohibition is violated (i.e. violating a prohibition means *doing* the prohibited action α). If we take this approach and give instead of the modality $O_{\mathcal{C}}(\alpha)$ only $O(\alpha)$ then we come into problems when coupling obligations with the exclusive choice operator $\oplus$.

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 strong recursion 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.

The symbol α denotes a compound action (i.e., an expression containing one or more of the following operators: choice "+"; sequence "$\cdot$"; concurrency "&", and test "?"). β are the actions found in dynamic logic which have the extra operator Kleene star *. The investigation of the restricted actions α and their interpretations as guarded rooted trees is caried out in Section 2. The investigation of the standard β actions can be found in the literature related to dynamic logic and to Kleene algebras; basically they define the regular sets and have the syntax of regular expressions.

[1] We take a rather general approach and consider any $\mathcal{CL}$ clause as a reparation. More natural would be when the reparation may be only contract clauses which are formed only of $O_{\mathcal{C}}$ and $F_{\mathcal{C}}$ expressions.

The Boolean operators $\wedge$ and $\oplus$ may be thought as the classical conjunction and exclusive disjunction, which may be used to combine obligations and permissions. For prohibition $\mathcal{C}_F$ we have $\vee$, again with the classical meaning of the corresponding operator. Note that syntactically $\oplus$ cannot appear between prohibitions.

We borrow from propositional dynamic logic [FL77] the syntax $[\beta]\mathcal{C}$ to represent that after performing β (if it is possible to do so), $\mathcal{C}$ must hold. The $[\cdot]$ notation allows having a *test*, where $[\mathcal{C}_1?]\mathcal{C}_2$ must be understood as $\mathcal{C}_1 \Rightarrow \mathcal{C}_2$. $\langle\beta\rangle\mathcal{C}$ captures the idea that it must exist the possibility of executing α, in which case $\mathcal{C}$ must hold afterwards.

Propositional dynamic logic can encode the temporal logic (TL) [Pnu77] operators $\mathcal{U}$ (*until*), $\bigcirc$ (*next*), $\Box$ (*always*), and $\Diamond$ (*eventually*). The special action $\mathbf{any} = a_1 + \ldots + a_n$ is a shortcat which stands for the finite choice between any basic action of $\mathcal{A}_B$ (see Section 2).[2] 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 [HTK00, sec.17.2] for more on embedding linear temporal logic operators into propositional dynamic logic.

Discussion: an 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 $\langle((\neg\varphi \wedge O(a \cdot b))? \cdot a \cdot b)^*\rangle\varphi$.

In $\mathcal{CL}$ we can write *conditional* obligations, permissions, and prohibitions of two different kinds. Just as an example let us consider conditional obliga-

[2]The definition of **any** depends on the semantics of the PDL modality $[\cdot]$. Consider that we interpret the semantics as in Section 3.1 where $[a]\mathcal{C}$ means that after an action γ which contains the a (i.e. $a \leq_\& \gamma$ and e.g. γ can be $a, a\&b, a\&b\&c$) is executed then $\mathcal{C}$ must hold. Therefore, the definition we give here for **any** is suficient for the purpose of encoding the TL next operator $\bigcirc$. Consider now that we interpret the $[\cdot]$ as in classical PDL as saying that $[a]\mathcal{C}$ means that after doing only a then $\mathcal{C}$ holds. In this case the definition of **any** changes to $\mathbf{any} = +_{\alpha_\& \in \mathcal{A}_B^\&} \alpha_\&$; i.e. the choice not only between basic actions but also choice between concurrent actions. See next section of the definitions of various new concepts like $\mathcal{A}_B^\&$.

tions. The first kind is represented as $[\beta]O(\alpha)$, which may be read as "after performing β, one is obliged to do α". The second kind is modeled using the test operator ?: $[\varphi?]O(\alpha)$, representing "If φ holds then one is obliged to perform α". Similarly for permission and prohibition. For convenience, in what follows we use the notation $\phi \Rightarrow \mathcal{C}$ instead of the $\mathcal{CL}$ syntax $[\phi?]\mathcal{C}$.

1.2 Examples

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 Algebra of Concurrent Actions

$\mathcal{CL}$ is heavily based on actions (i.e. all modal operators are defined over actions). We have investigated in [PS07a] the actions found in contracts and formalized them as an algebra of actions. In this section we give a slightly different algebraic structure but the main results, discussions, examples, or background information are the same as in [PS07a]. For in depth reading and for a more clear and motivated presentation we refer the reader to [PS07a]. Here we merely discuss the differences.

A main change is that the concurrency operator & is idempotent; thus no longer having multisets as labels (but sets) and we can no longer represent discrete values as we did with the non-idempotent version of &. Throughout this section we make more clear these ideas, and also present briefly all the other features of the algebra of actions found in contracts.[3]

Though the algebraic structure we define is somehow similar to Kleene algebra with tests [Koz97], there are substantial differences due mainly to our application domain. A first difference is that we do not include the Kleene star (iteration) as it is not needed in our context[4] (see [PS07b]). A second

[3]Actions can be enriched with features like *parameters*, *subjects*, or *durations*; this is the topic of a follow-up report. To compensate the loss of expresivity of the actions we can add parameters of type $\mathbb{N}$ (`Nat`) so that we can again express discrete quantities. This is just an example of adding parameters to the actions (the parameters may be of any type with the operations and reasoning associated). On the same ideas actions can be enriched with subjects or durations.

[4]It is counter intuitive to have iteration of actions under obligation, permission, or prohibition; e.g. it is not normal to have in a contract a statement like: "One is obliged

difference is that we introduce an operator to model concurrency. The algebra has the following particularities: (1) Formalizes concurrent actions; (2) Introduces a different kind of negation over actions; and (3) Has a standard interpretation of the actions over specially defined rooted trees. Among others, the interpretation using trees is used for giving the branching semantics of Section 3.1.

2.1 The algebraic structure $\mathcal{CA}$

We start by defining an algebraic structure $\mathcal{CA} = (\mathcal{A}, \Sigma)$ which is the basis of the *algebra of concurrent actions and tests* $\mathcal{CAT} = (\mathcal{CA}, \mathcal{B})$ presented in this section. $\mathcal{CA}$ defines the concurrent actions, and the Boolean algebra $\mathcal{B}$ of Section 2.3 defines the tests.

The algebraic structure $\mathcal{CA}$ is defined by a carrier set of elements (called *compound actions*, or just actions) denoted $\mathcal{A}$ and by the signature $\Sigma = \{+, \cdot, \&, \mathbf{0}, \mathbf{1}, \mathcal{A}_B\}$ which gives the action operators and the basic actions. More precisely $\mathcal{CA}$ is a family of algebras indexed by the finite set of basic (or atomic) actions $\mathcal{A}_B$. The non-constant functions of Σ are: "+" for *choice* of two actions, "$\cdot$" for *sequence* of actions (or concatenation), and "&" for *concurrent* composition of two actions. Each of the operators $+$, $\cdot$, and $\&$ takes two actions and generates another action of $\mathcal{A}$. The special elements $\mathbf{0}$ and $\mathbf{1}$ are constant function symbols. The set $\mathcal{A}_B \cup \{\mathbf{0}, \mathbf{1}\}$ is called the *generator set* of the algebra. The basic actions of $\mathcal{A}_B$ and $\mathbf{0}$ and $\mathbf{1}$ have the property that cannot be generated from other actions of $\mathcal{A}$.

To be more precise about the syntactic structure of the actions of $\mathcal{A}$ we set the rules for constructing *actions*. The operators $+$, $\cdot$, and $\&$ are sometimes called *constructors* because they are used to construct all the actions of $\mathcal{A}$ as we see in Definition 2.1. This defines the term algebra $T_{\mathcal{CA}}(\mathcal{A}_B)$ parameterized by the set of basic actions $\mathcal{A}_B$ which is free in the corresponding class of algebras over the generators of $\mathcal{A}_B \cup \{\mathbf{0}, \mathbf{1}\}$. We will just use $T_{\mathcal{CA}}$ whenever $\mathcal{A}_B$ is understood by context.

Definition 2.1 (action terms of $\mathcal{CA}$)**.**

1. *any basic action a of $\mathcal{A}_B$ is an action of $\mathcal{A}$;*
2. *$\mathbf{0}$ and $\mathbf{1}$ are actions of $\mathcal{A}$;*
3. *if $\alpha, \beta \in \mathcal{A}$ then $\alpha\&\beta$, $\alpha \cdot \beta$, and $\alpha + \beta$ are actions of $\mathcal{A}$;*
4. *nothing else is an action of $\mathcal{A}$.*

to not pay, or pay once, o pay twice, or ...".

(1)	$\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$	(10)	$\alpha\&(\beta\&\gamma) = (\alpha\&\beta)\&\gamma$
(2)	$\alpha + \beta = \beta + \alpha$	(11)	$\alpha\&\beta = \beta\&\alpha$
(3)	$\alpha + \mathbf{0} = \mathbf{0} + \alpha = \alpha$	(12)	$\alpha\&\mathbf{1} = \mathbf{1}\&\alpha = \alpha$
(4)	$\alpha + \alpha = \alpha$	(13)	$\alpha\&\mathbf{0} = \mathbf{0}\&\alpha = \mathbf{0}$
(5)	$\alpha \cdot (\beta \cdot \gamma) = (\alpha \cdot \beta) \cdot \gamma$	(14)	$a\&a = a$ for $a \in \mathcal{A}_B$
(6)	$\alpha \cdot \mathbf{1} = \mathbf{1} \cdot \alpha = \alpha$	(15)	$\alpha\&(\beta + \gamma) = \alpha\&\beta + \alpha\&\gamma$
(7)	$\alpha \cdot \mathbf{0} = \mathbf{0} \cdot \alpha = \mathbf{0}$	(16)	$(\alpha + \beta)\&\gamma = \alpha\&\gamma + \beta\&\gamma$
(8)	$\alpha \cdot (\beta + \gamma) = \alpha \cdot \beta + \alpha \cdot \gamma$	(17)	$\alpha\&(\alpha' \cdot \beta) \rightarrow \alpha(1)\&\alpha'(1) \cdot \ldots \cdot \alpha(n)\&\alpha'(n) \cdot \beta$
(9)	$(\alpha + \beta) \cdot \gamma = \alpha \cdot \gamma + \beta \cdot \gamma$		where $l(\alpha) = l(\alpha') = n$

Table 3: Axioms of $\mathcal{CA}$

Throughout this report we denote by $a, b, c, \ldots$ elements of $\mathcal{A}_B$ (basic actions) and by $\alpha, \beta, \gamma, \ldots$ elements of $\mathcal{A}$ (compound actions). When the difference between basic and compound actions is not important we just call them generically *actions*. For brevity we often drop the sequence operator and instead of $\alpha \cdot \beta$ we write $\alpha\beta$. To avoid unnecessary parentheses we use the following precedence over the constructors: $\& > \cdot > +$.

To have a complete algebraic theory we include the two special elements $\mathbf{0}$ and $\mathbf{1}$ which are the identity elements for $+$, respectively for $\cdot$ and $\&$ operators. We call action $\mathbf{1}$ the *skip* action and $\mathbf{0}$ the *violating* action. In Table 3 we collect the axioms that define the structure $\mathcal{CA}$.

The properties of the operators $+$ and $\cdot$ are defined by the axioms (1)-(9) of Table 3. Axioms (1)-(4) define $+$ to be associative, commutative, with identity element $\mathbf{0}$, and idempotent. Axioms (5)-(7) define $\cdot$ to be associative, with identity element $\mathbf{1}$, and with annihilator $\mathbf{0}$. The element $\mathbf{0}$ is an annihilator for the sequence operator both on the left and on the right side. We call the two equations *fail late* (for $\alpha \cdot \mathbf{0} = \mathbf{0}$) and *fail soon* (for $\mathbf{0} \cdot \alpha = \mathbf{0}$). Axioms (8)-(9) give the distributivity of $\cdot$ over $+$; property which we exploit more in Section 2.5 when we define a *canonic form of actions*. Because the $+$ operator is idempotent ($\alpha + \alpha = \alpha$) all these axioms give the algebraic structure of an idempotent semiring $(\mathcal{A}, +, \cdot, \mathbf{0}, \mathbf{1})$.

The third constructor $\&$ is intended to model, what we call, *eager true concurrency*. At this point we give an informal intuition of the elements (actions) of $\mathcal{A}$: we consider that the actions are performed by somebody (being that a person, a program, or an agent). We talk about "performing" and one should not think of *processes executing actions* and operational semantics; we do not discuss operational semantics in this report. With this non-algebraic intuition of actions we can elaborate on the purpose of $\&$, which models the fact that two actions are performed in a truly concurrent fashion. We call

concurrent actions and denote by $\mathcal{A}_B^{\&} \subseteq \mathcal{A}$ the subset of elements of $\mathcal{A}$ generated using only & constructor (e.g. a, $a\&b \in \mathcal{A}_B^{\&}$ and $a+b, a\&b+c, a\cdot b \notin \mathcal{A}_B^{\&}$). Note that $\mathcal{A}_B^{\&}$ is finite because there is a finite number of basic actions in $\mathcal{A}_B$ which may be combined with the concurrency operator & in a finite number of ways (due to the idempotence of & over basic actions; axiom (14)).

Axioms (10)-(13) give the properties of & to be associative, commutative, with identity element $\mathbf{1}$, annihilator element $\mathbf{0}$ which make the algebraic structure $(\mathcal{A}, \&, \mathbf{1})$ commutative monoid with element $\mathbf{0}$ as *annihilator* for &. Axioms (10) and (11) basically say that the syntactic ordering of actions in a concurrent action does not matter (the same as for choice +). Axiom (14) defined & to be idempotent over the basic actions $a \in \mathcal{A}_B$.[5] Axioms (15) and (16) define the distributivity of & over +. From axioms (10)-(16) together with the fact that $(\mathcal{A}, +, \mathbf{0})$ is a commutative monoid we may conclude that $(\mathcal{A}, +, \&, \mathbf{0}, \mathbf{1})$ is a commutative and idempotent semiring.[6]

For *Example 1.* in the introduction the basic actions are $\mathcal{A}_B = \{ebl, p, ne, d\}$ (for "extend bandwidth limit", "pay", "notify by email", and "delay"). An example of a concurrent action is $d\&ne \in \mathcal{A}_B^{\&}$, or even $ebl \,\&\, p \,\&\, d \,\&\, ne$.

Note that throughout this section we use well known notions like strings, sets, or multisets in association with our actions just for presentation purposes only. All definitions or explanations (e.g. Definition 2.3) using these classical notions can be given in a purely syntactical manner.

For axiom (17) we need some preliminary notions introduced in the following. We consider that basic actions are instantaneous[7] with regard to their execution time and we introduce the notion of *length of an action*.

Definition 2.2 (action length)**.**

The length of an action α *is defined (inductively) as a function* $l : \mathcal{A} \to \mathbb{N}$ *which takes as argument an action and returns a natural number.*

1. $l(\mathbf{1}) = l(\mathbf{0}) = 0$
2. $l(a) = 1$ *for any basic action* a *of* $\mathcal{A}_B$,
3. $l(\alpha\&\beta) = l(\alpha + \beta) = max(l(\alpha), l(\beta))$,
4. $l(\alpha \cdot \beta) = l(\alpha) + l(\beta)$.

[5] Note that this does *not* imply that we have an idempotent monoid.

[6] $(\mathcal{A}, +, \&, \mathbf{0}, \mathbf{1})$ is an idempotent semiring because the first operator + is idempotent; and not because of the weaker idempotency property of & which is defined only over basic actions.

[7] Note that when actions are extended with durations (i.e. the amount of time necessary for the action to finish) some of these notions (like action length) need to be changed accordingly.

$max : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ is the standard function returning the maximum value of the two arguments. For the special actions $\mathbf{1}$ and $\mathbf{0}$ the length is 0. The intuition of the length function is that it counts the number of actions in a sequence of actions given by the $\cdot$ constructor. From this perspective we view the compound actions as strings where the elements of the string are separated by the sequence constructor. We say that $\alpha(n)$ identifies the action on position n in the string of actions α. The position $0 < n \leq l(\alpha)$ is a strictly positive integer less than or equal to the length of the action. For $n = 0$, $\alpha(0) = \mathbf{1}$ returns the implicit *skip* action, which is natural because every action α can have as starting action $\mathbf{1}$, i.e. $\alpha = \mathbf{1} \cdot \alpha$. For example, for action $\alpha = (a + b) \cdot \mathbf{1} \cdot c$ we have $l(\alpha) = 2$, $\alpha(1) = a + b$ and $\alpha(2) = c$. Note that $\alpha(\cdot)$ ignores $\mathbf{1}$'s. Therefore, $\mathbf{1}$ is ignored in the axiom (17) because $\mathbf{1}$ can be removed from a sequence as it is the identity element for $\cdot$ operator; e.g. an action $a\&(\mathbf{1} \cdot b)$ is equivalent to $a\&b$.

Specific to our application domain we consider it is natural to relate & and $\cdot$ as follows.

$$\text{if } l(\alpha){=}l(\alpha'){=}n \text{ then } \alpha\&(\alpha' \cdot \beta) = \alpha(1)\&\alpha'(1) \cdot \ldots \cdot \alpha(n)\&\alpha'(n) \cdot \beta \quad (17)$$

A similar equation can be given for the corresponding action $(\alpha' \cdot \beta)\&\alpha$ with the sequence on the left side of the concurrency constructor. The equation is defined using the action length. One can define the same property using sorts as: $(\alpha_{\&} \cdot \alpha)\&(\beta_{\&} \cdot \beta) = \alpha_{\&}\&\beta_{\&} \cdot \alpha\&\beta$ where $\alpha_{\&}, \beta_{\&} \in \mathcal{A}_B^{\&}$ are only concurrent actions and α, β can be any kind of action. We prefer the one using action length as it is more clear and explicit.

We call the notion modelled by axiom (17) *eager true concurrency*. Intuitively that is because if we execute concurrently two sequences of actions the behavior (which we want) is to execute *immediately* the first action of the first sequence and the first action of the second sequence concurrently and only after these are finished continue with executing the remaining actions.

Let us take a look at the properties of + and & of being idempotent. If we take compound actions constructed only with + then because of the idempotence we do not find the same basic action twice in the compound action. For example, action $a + a + b$ is the same as $a + b$ after we apply the idempotence equation. From this point of view we consider that the basic actions of an additive compound action (i.e. a compound action generated only with +) form a *set* included in $\mathcal{A}_B$. The same holds for &; executing an action concurrently with another copy of itself is impossible. We denote by $\{\alpha_+\}$ the set of basic actions associated to a compound action α_+ which is constructed only with +; similarly we denote by $\{\alpha_{\&}\}$ the set of basic actions associated to a compound concurrent action $\alpha_{\&}$ which is constructed

only with &. When it is obvious from the context that we ar talking about the set of basic actions associated to an action $\alpha_{\&}$ and not about the action itself we use just $\alpha_{\&}$ instead of $\{\alpha_{\&}\}$.

Pratt [Pra86] introduces the concept of partially ordered multisets (or *pomsets*) to model truly concurrent processes; i.e. processes which are sequences of events denoting actions. Pratt's theory reasons about complex systems and (time) ordering of the actions of processes, which is too powerful for our purpose. We do not want to model entire processes that are truly concurrent, and we do not need true concurrency over time periods because we do not have any notion of time in our model. For now we only want to model atomic actions executing in a truly concurrent fashion.

Note that for our purpose the approach of considering the concurrent actions as sets over the basic actions is in the spirit of Pratt's theory. A pomset intuitively states that if two events labelled by some actions are related by the partial order of the pomset then the events are not concurrent, but are executed in the sequence given by their ordering. On the other hand, any events that are not related by the partial order are considered truly concurrent. We recall that a set is equivalent to a pomset with the empty order as the partial order (and an additional condition of injective labelling). The empty order intuitively means that no event is related to another, which in the theory of pomsets means that all the events of the set are executed concurrently.

With the view of concurrent actions as sets over $\mathcal{A}_B$ we can define a strict partial order over concurrent actions with the help of inclusion of sets.

Definition 2.3 (demanding relation)**.**
We define the relation $<_{\&}$ as:

$$\alpha_{\&} <_{\&} \beta_{\&} \stackrel{def}{=} \{\alpha_{\&}\} \subset \{\beta_{\&}\} \tag{18}$$

where $\alpha_{\&}, \beta_{\&} \in \mathcal{A}_B^{\&}$ are concurrent actions, and $\{\alpha_{\&}\}$ denotes the set of basic actions associated to $\alpha_{\&}$.

We call $<_{\&}$ the *demanding relation* with the intuition that β is more demanding than α iff $\alpha <_{\&} \beta$. We consider the action $\mathbf{1}$ as the empty set, with the intuition that skiping means not doing any action. Note that the least demanding action is $\mathbf{1}$. 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 $<_{\&}$. We denote by $\leq_{\&}$ the relation $<_{\&} \cup =$; i.e. $\alpha \leq_{\&} \beta$ iff either $\alpha <_{\&} \beta$ or $\alpha = \beta$.

Proposition 2.1 ([PS07a])**.** *The relation $<_{\&}$ is a strict partial order.*

For a better intuitive understanding take the following examples: $\mathbf{1} <_\& a$, $a <_\& a\&b$, $a\&b <_\& a\&c\&b$, $a \not<_\& b$, $a \not<_\& a$, and $a \not<_\& b\&c$.

By now we have defined the demanding relation only on concurrent compound actions (i.e. for actions of the form $\alpha = a_1\& \ldots \& a_n$). In order to extend $<_\&$ to the whole carrier set $\mathcal{A}$ we need to extend the definition with sets for the & to some more complex definitions for $\cdot$ and $+$ (see [PS07a]).

Because $+$ is idempotent we can still define as in Kleene algebra a partial order $\geq^+$ on the elements of $\mathcal{A}$. We call it the *preference relation*. That is: $\alpha \geq^+ \beta$ means that action α has higher preference over action β.

Definition 2.4 (preference relation)**.** *The preference relation is defined as:*

$$\alpha \geq^+ \beta \iff^{def} \alpha + \beta = \alpha \tag{19}$$

An intuition for this is that whenever one has to choose among the two actions α and β one always chooses α, the most preferable action (i.e. $\alpha+\beta = \alpha$). Note that $\mathbf{0}$ is the least preferable action because $\mathbf{0} + \alpha = \alpha$; so $\mathbf{0}$ is never preferred over another action different than itself.

Proposition 2.2 ([PS07a])**.** *The preference relation $\geq^+$ is a partial order.*

Note that the three operators are monotone with respect to the partial order $\geq^+$. For example for any actions α, β, and γ, for $+$ operator this means that: *if* $\alpha \geq^+ \beta$ *then* $\alpha + \gamma \geq^+ \beta + \gamma$. This is easily proven.

Discussion: The *intersection* operator $\cap$ over actions form PDL$^\cap$ (i.e. PDL extended with intersection of actions; see [HTK00, 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 & 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 & may have unwanted behavior because of the combination of idempotency and distributivety. For example the rewriting system associated is not confluent; which means that one term may rewrite to two different terms. For example: $(a + b)\&(a + b) \overset{idem}{=} a + b$ and also $(a + b)\&(a + b) \overset{dist}{=} a\&a + a\&b + a\&b + b\&b \overset{idem}{=} a + a\&b + b$, and the two

terms on the right are not the same. One might argue that the two different terms when used inside any of our $\mathcal{CL}$ modalities yeld the same models (i.e. are interpreted the same), so why to make a difference between them?

In the case of intersection $\cap$ this problem does not occure because of another property intersection has: *absorbtion*. Absorbtion property 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 $\alpha <_{\&} \beta$ then $\alpha\&\beta = \alpha$. This means that when choosing between $a + (a\&b)$ 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.

We can get a confluent rewriting system if we restrict the idempotence of & only to atomic actions, as in axiom (14). This restriction does not influence the desired behavior for the actions as by the rules that we have the & operator is first pushed inside towards the basic actions and only then the idempotence is applied.

Note that combination of compound actions $\alpha\&\alpha\&\alpha\&\ldots$ cannot go indefinitely as it reaches a fixpoint. For example $(a+b)\&(a+b)\&(a+b)\&\ldots = (a+b)\&(a+b)$ can be easily checked by induction where the basic case is $(a+b)\&(a+b)\&(a+b) = (a+b)\&(a+b)$. The right hand side is equal to $a + a\&b + b$ and the left hand side is equal to $(a + a\&b + b)(a + b) = a + a\&b + a\&b + a\&b + a\&b + b = a + a\&b + b$.

We define a relation for the concurrency operator & similar to the preference relation $\geq^{+}$. We call it the *absorbtion relation*[8] and denote it by $<^{ab}_{\&}$.

Definition 2.5 (absorbtion relation)**.** *The absorbtion relation $<^{ab}_{\&}$ is defined over compound actions of $\mathcal{CA}$ as:*

$$\alpha <^{ab}_{\&} \beta \text{ iff } \alpha\&\beta = \beta$$

For the absorbtion relation we can prove the following properties:

[8]Not to be confused with the absorbtion property.

1. transitivity: for all α, β, γ if $\alpha <^{ab}_{\&} \beta$ and $\beta <^{ab}_{\&} \gamma$ then $\alpha <^{ab}_{\&} \gamma$;

   Proof: From $\alpha <^{ab}_{\&} \beta$ and $\beta <^{ab}_{\&} \gamma$ we have $\alpha\&\beta = \beta$ *(i)* and $\beta\&\gamma = \gamma$ *(ii)*, and we need to prove $\alpha\&\gamma = \gamma$. We have the following $\alpha\&\gamma \overset{(ii)}{=} \alpha\&\beta\&\gamma \overset{(i)}{=} \beta\&\gamma \overset{(ii)}{=} \gamma$. □

2. antisymmetry: for all α, β, if $\alpha <^{ab}_{\&} \beta$ and $\beta <^{ab}_{\&} \alpha$ then $\alpha = \beta$ (α and β are the same element).

   Proof: From the first inequality we have by definition that $\alpha\&\beta = \beta$. By commutativity of & we get that $\beta\&\alpha = \beta$ which by the second inequality we get $\beta\&\alpha = \alpha$, and thus $\alpha = \beta$. □

3. reflexivity: for all α we have $\alpha <^{ab}_{\&} \alpha$;

   Proof: This is not true for all compound actions because the operator & is not idempotent over all actions. Nevertheless, $<^{ab}_{\&}$ is reflexive over some particular actions. It is reflexive over basic actions of $\mathcal{A}_B$ because of the idempotence axiom (14).

   For example an action like $\alpha = (a+b)$, taken from the discussion above, breaks the reflexivity because as shown above $(a+b)\&(a+b) \neq (a+b)$.

   On the other hand there are other compound actions for which reflexivity holds. One example of actions comes from the discussion above. Take $\beta = (a+b)\&(a+b)$; then we have $\beta\&\beta = \beta$ because $\beta\&\beta = (a+b)\&(a+b)\&(a+b)\&(a+b) = (a+b)\&(a+b)\&(a+b) = (a+b)\&(a+b) = \beta$. □

We consider a relation over the set of basic actions $\mathcal{A}_B$ which we call *conflict relation* and denote by $\#_{\mathcal{C}}$. The intuition of the conflict relation is that if two actions are in conflict then the actions cannot be executed concurrently. This relation is defined in terms of the & operator and says that two actions that are in conflict, when executed concurrently yield the special action $\mathbf{0}$. The converse relation of $\#_{\mathcal{C}}$ is the *compatibility relation* which we denote by $\sim_{\mathcal{C}}$. The intuition of the compatibility relation is that if two actions are compatible then the actions can be executed concurrently.

Definition 2.6 (conflict and compatibility)**.**

The conflict relation is defined as:

$$a \#_{\mathcal{C}} b \overset{def}{\iff} a\&b = \mathbf{0} \tag{20}$$

The compatibility relation is defined as:

$$a \sim_{\mathcal{C}} b \overset{def}{\iff} a\&b \neq \mathbf{0}, \text{ where } a, b \neq \mathbf{0} \tag{21}$$

Proposition 2.3 ([PS07a]). *The following standard properties of the conflict and compatibility relations for basic actions hold: reflexivity and symmetry of $\sim_{\mathcal{C}}$, and symmetry of $\#_{\mathcal{C}}$.*

Remark: There is *NO transitivity* of $\#_{\mathcal{C}}$ or $\sim_{\mathcal{C}}$: In general, if $a \#_{\mathcal{C}} b$ and $b \#_{\mathcal{C}} c$, not necessarily $a \#_{\mathcal{C}} c$. This is natural as action b may be in conflict with both a and c but still $a \sim_{\mathcal{C}} c$.

The definition of the conflict and compatibility relations extend to all actions of $\mathcal{A}$ by extending $\#_{\mathcal{C}}$ and $\sim_{\mathcal{C}}$ to the $+$, $\cdot$, and $\&$ operators; see [PS07a].

2.2 Standard interpretation of $\mathcal{CA}$ over rooted trees

We give the standard interpretation of the actions of $\mathcal{A}$ by defining a homomorphism $I_{\mathcal{CA}}$ which takes any action of the $\mathcal{CA}$ algebra into a corresponding rooted tree and preserves the structure of the action given by the constructors. Before this, we define what are *rooted trees* and the operations we consider over them.

2.2.1 Rooted trees

In this section we give the definition of *rooted trees* and define several operations over rooted trees.

Definition 2.7 (rooted tree). *A rooted tree is an acyclic connected graph $(\mathcal{N}, E)$ with a designated node r called root node. $\mathcal{N}$ is the set of nodes and E is the set of edges (where an edge is a pair of nodes (n, m)).*

An alternative definition of trees comes from ordered sets theory: a *rooted tree* is a partially ordered set $(\mathcal{N}, <)$ of nodes such that for each node $n \in \mathcal{N}$ all the nodes $m \in \mathcal{N}$ less than n with respect to the order $<$ (i.e. $m < n$) are well-ordered[9] by the relation $<$, and there is only one least element r called the root node. In this definition the nodes m are called the ancestor nodes of node n, and their property of being well-ordered gives the intuitive property of nodes in a tree (except the root node) to have one and only one *parent*[10] node. Because of the partial order on the nodes of the tree we consider that we have *directed edges* (i.e. the tree is a special *directed graph*), with the direction of the edges going from the root node to the higher nodes with

[9]The well-ordering of the set $N = \{m \,|\, m < n\}$ with respect to the partial order $<$ means that the partial order $<$ transforms into a *total order* on N and for each subset $S \subset N$ there exists a least element with respect to the total order.

[10]A node m is the parent of node n iff $m < n$ and $\nexists k \in \mathcal{N}$ s.t. $m < k$ and $k < n$.

respect to the partial order. Note that there cannot be two edges (n, m) and (m, n) in the same tree. All nodes $\{m \mid (n, m)\}$ are called the descendents (or children) nodes of n. The *siblings* of a node m are all the nodes which have common parent with m; i.e. $sibl(m) = \{m' \mid (n, m), (n, m') \in E\}$. Note that the root node has no siblings.

We consider rooted trees with labeled edges; i.e. each edge (n, m) has associated a label α. We denote the labeled directed edges of the tree with (n, α, m) and the tree with $(\mathcal{N}, E, \mathcal{A})$. The labels $\alpha \in \mathcal{A}$ are sets of *basic labels* of $\mathcal{A}_B \in \mathcal{A}$; e.g. $\alpha_1 = \{a, b\}$ or $\alpha_2 = \{a\}$ with $a, b \in \mathcal{A}_B$. For the sake of notation we use a instead of the singleton set $\{a\}$. Comparing two labels α and β for equality means comparing the two associated sets. We denote by τ the special empty label that is the empty set. When the label is not important (i.e. can be any label) we may use the notation (n, m) instead of (n, α, m) $\forall \alpha \in \mathcal{A}$.

We restrict our presentation to *finite* rooted trees. This means that there is no infinite chain of nodes $r < n_1 < n_2 \ldots$ (or equivalently, there is no infinite path in the directed graph starting from the root node). Such chains are called *branches* of the tree. The final nodes on each branch are called *leaf nodes*. The *height* of a tree T denoted $h(T)$ is the number of edges in the longest branch of the tree which are not labelled by τ.

Definition 2.8 (Tree isomorphism)**.** *Two trees $T_1 = (\mathcal{N}_1, E_1, \mathcal{A}_1)$ and $T_2 = (\mathcal{N}_2, E_2, \mathcal{A}_2)$ are isomorphic, denoted $T_1 \doteq T_2$, iff $\mathcal{A}_1 = \mathcal{A}_2$ (the labels are the same), and there is a bijective function $rn : \mathcal{N}_1 \to \mathcal{N}_2$ s.t. $rn(root_1) = root_2$ and $\forall (n, \alpha, m) \in E_1$ then $(rn(n), \alpha, rn(m)) \in E_2$.*

Equivalently, we say that the relation $\doteq$ denotes the *equality modulo renaming of the nodes* between two rooted trees. Besides modulo renaming of the nodes the relation $\doteq$ is based on the usual equality on rooted trees where for example the branches of a tree are not ordered.

Examples of 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;

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.

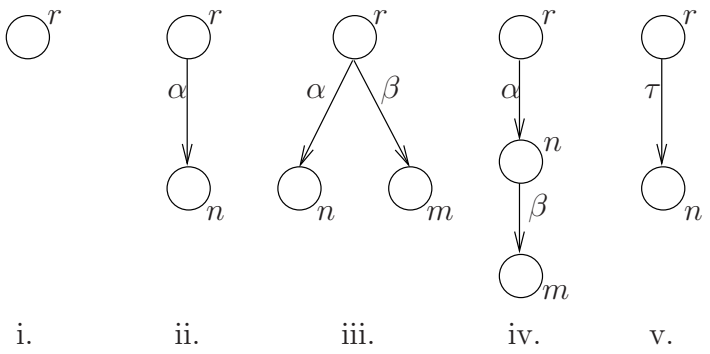

Figure 1: Examples of finite rooted trees with labeled edges.

In the following we define three operations on rooted trees. We consider the classical notion of *subtree*. The first operation is the *join* of two trees and we denote it by $\cup$. Take two trees $T_1 = (\mathcal{N}_1, E_1, \mathcal{A}_1)$ with root r_1 and $T_2 = (\mathcal{N}_2, E_2, \mathcal{A}_2)$ with root r_2 as in Figure2. Note that the two sets of nodes are disjoint (thus also the sets of edges are disjoint), where the two sets of labels may have elements in common. Joining T_1 and T_2 consists in the following steps:

1. join the two root nodes r_1 and r_2 into a single root node (call it r_{12});

2. make the union of the two sets of nodes $\mathcal{N}_{12} = \mathcal{N}_1 \setminus \{r_1\} \cup \mathcal{N}_2 \setminus \{r_2\} \cup \{r_{12}\}$, and the union of the two label sets $\mathcal{A}_{12} = \mathcal{A}_1 \cup \mathcal{A}_2$;

3. add to the empty set of edges E_{12} the edges on the first level of the two trees, i.e. $E_{12} = \{(r_{12}, n) \,|\, (r_1, n) \in E_1\} \cup \{(r_{12}, m) \,|\, (r_2, m) \in E_2\}$;

4. for each two edges in E_{12} labeled with the same label ((r_{12}, α, n) and (r_{12}, α, m)) keep only one edge in E_{12} and do the same joining operation for the subtrees with roots n respectively m (in the case when one of the subtrees has only the root node n and the other has several edges then consider the tree with only one root node as the expanded tree with only one edge $(n, \tau, n' : W)$). For all other single edges (r_{12}, k) just add to E_{12} all the edges of the subtrees with the root node k.

Note that the height of the new tree is the maximum of the heights of the two joined trees: if we have $h(T_1)$ and $h(T_2)$ then $h(T_{12}) = max(h(T_1), h(T_2))$.

The second operation is the *concatenation* of two trees and we denote it by $\widehat{\ }$. Take the two trees T_1 and T_2 as before. The picture in Figure3 illustrates this operation. To concatenate T_1 with T_2 follow the steps:

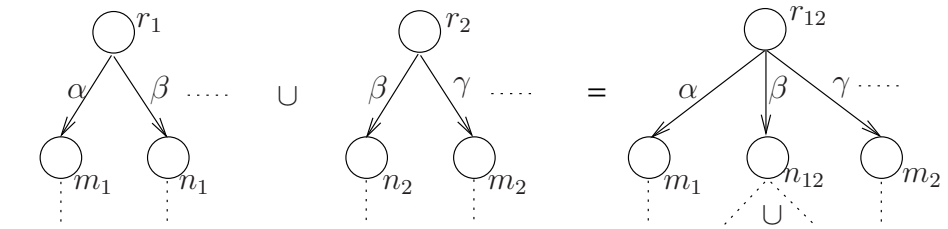

Figure 2: Join of two rooted trees.

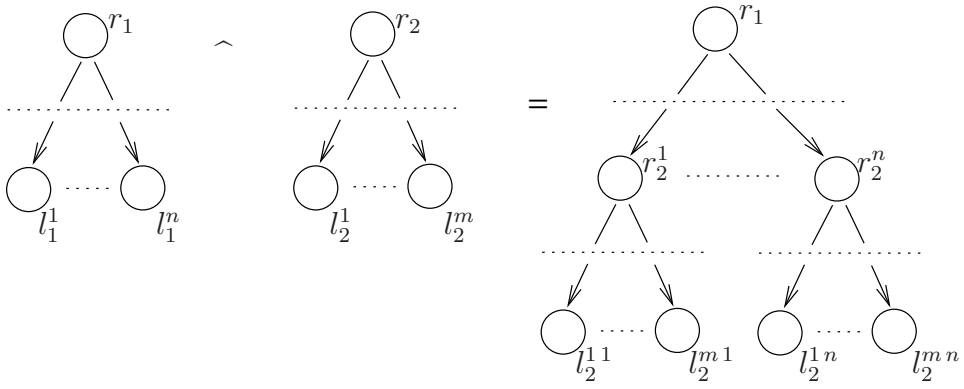

Figure 3: Concatenation of two rooted trees.

1. take the resulting tree T_{12} to be T_1 for start. That means that $\mathcal{N}_{12} = \mathcal{N}_1$, $E_{12} = E_1$, and $\mathcal{A}_{12} = \mathcal{A}_1$.

2. replace each of the leaf nodes of T_{12} with the whole tree T_2. This means:

   (a) replace each edge (n, m) with node m a leaf node of T_{12} with (n, r_2);

   (b) remove each leaf node from $\mathcal{N}_{12}$;

   (c) add all the nodes of T_2 to $\mathcal{N}_{12}$ renaming them such that each node in $\mathcal{N}_{12}$ has a different name;

   (d) add all the edges of E_2 to E_{12} with the nodes names changed accordingly to step 2c.

After the concatenation operation the new tree T_{12} has the hight equal to the sum of the heights of the two trees: $h(T_{12}) = h(T_1) + h(T_2)$.

A third operation over our rooted trees is the *concurrent join* which we denote by $\|$. Concurrent joining involves also manipulating labels (basically union and comparison of sets). The procedure of concurrently joining two trees T_1 and T_2 taken as before consists in the following steps:

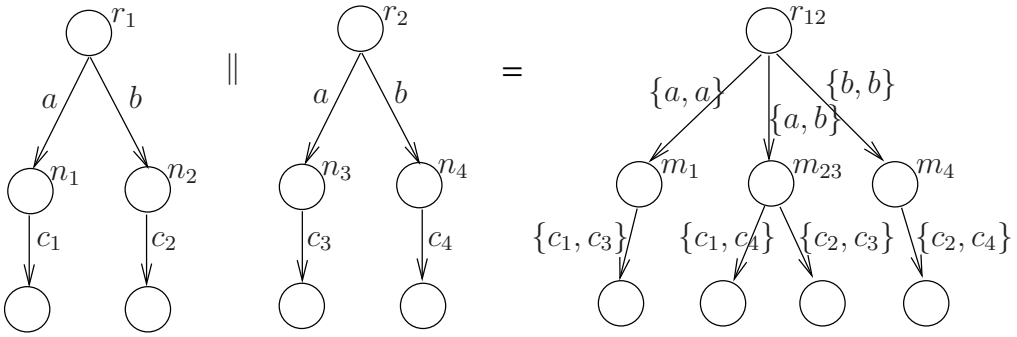

Figure 4: Example of concurrent join of two rooted trees.

1. join the two root nodes r_1 and r_2 into a single root node and call it r_{12};
2. take the edges on the first level of each tree $\{(r_{12}, \alpha_1, n_1) \,|\, (r_1, \alpha_1, n_1) \in E_1\}$ and $\{(r_{12}, \alpha_2, n_2) \,|\, (r_2, \alpha_2, n_2) \in E_2\}$ and combine them as follows:
   (a) combine the labels α_i two by two.[11] Each new label $\alpha' = \alpha_1 \cup \alpha_2$ is the set union of the two component labels.
   (b) add a new edge (r_{12}, α', m) to E_{12} and add to $\mathcal{N}_{12}$ the new node m obtained by joining n_1 and n_2;
   (c) for each two edges (r_{12}, α_1, n_1) and (r_{12}, α_2, n_2) of the old edge sets combined as in step (a) continue recursively to concurrently join the two subtrees with the roots in the nodes n_1 and n_2 and put the root of the new tree in the new node m created in step 2b.
   (d) for each two new edges (r_{12}, α', m_1) and (r_{12}, α'', m_2) of E_{12}, if $\alpha' = \alpha''$ then unify the two edges into one and make the *join* of the two new trees with roots in m_1 and m_2 created in step (c).

The height of the new tree is the maximum of the heights of the two combined trees. This is because none of the steps (a)-(c) do not add to the height of the new tree, and also the join in step (d) preserves the height. An example of concurrent joining of two trees is given in Figure 4.

In the $\|$ operation we use the union of two labels which is the union of the two associated sets. Note that the empty label τ "vanishes" when is joined with another label because τ is the empty set and $\emptyset \cup \{\ldots\} = \{\ldots\}$. We sometimes abuse the notation and instead of the union of two labels $\alpha \cup \beta$ (as they are considered sets of basic labels) we just write $\{\alpha, \beta\}$. Moreover, the τ label is often omitted so we consider $\{\tau, \alpha, \beta\} = \{\alpha, \beta\}$.

[11] As in a cartesian product of two sets.

For our purpose of giving a standard interpretation for $\mathcal{CA}$ we need to be able to interpret the special actions **1** and **0**, and therefore we make our rooted trees more particular. Each tree has two kinds of nodes that we distinguish by colors: the normal nodes (we have seen until now) are called *white nodes* and the new kind of special nodes are called *black nodes*. The black nodes are treated different (as we see below) and are found seldom in a tree. Note that the operations on trees must preserve the colors of the nodes. We sometimes use the notation $n : B$ and $n : W$ to denote the fact that node n is black or white respectively. The exact use of black nodes will become clear shortly.

Let us denote by $\mathcal{RT}$ the set of rooted trees. All the rooted trees in this set are created from a set of minimal trees using the operations *join*, *concatenation*, and *concurrent join* that we have defined in this section. The set of *minimal rooted trees* is denoted by $\mathcal{RT}_B$ and contains the trees formed only of one root node, and the trees with only one edge labeled with a basic label of $\mathcal{A}_B$ or τ. Thus the number of basic trees is $|\mathcal{RT}_B| = |\mathcal{A}_B| + 2$.

We give a normalization technique called *pruning a tree* which refers mostly to the empty label τ and to the *black nodes*.

Definition 2.9 (pruned tree)**.** *A pruned tree is a rooted tree obtained from any rooted tree with black and white nodes and τ edges by applying the four steps of the procedure below in that specific order.*

1. *contract all the τ labels on each path as follows:*
   (a) *for sets $\{\tau, \alpha, \ldots\}$ the τ "vanishes", i.e. we write the label $\{\alpha, \ldots\}$;*
   (b) *for all edges $(m, \tau, n : W)$ labelled with τ s.t. n is not a leaf node with siblings, remove the edge (and combine the two nodes of the endge into one) unless 1c;*
   (c) *if the tree has only one edge (r, τ, n) then do nothing;*
2. *for each* black *node n:*
   (a) *first remove the subtree with root n;*
   (b) *afterwards label the edge (m, α, n) with τ, where α is an arbitrary label;*
3. *for each edge (m, τ, n) with n a* black *node do either of:*
   (a) *if $\nexists (m, \alpha, n')$ a different edge with an arbitrary label α then remove the one edge before m, i.e. remove (k, β, m);*

(b) *if* $\exists(m, \alpha, n')$ *a different edge with an arbitrary label* α *then remove* (m, τ, n)*;*

4. *repeat step 3 as long as possible.*

The above procedure refers mostly to the empty label τ and to the black nodes. Consider the set $\mathcal{RT}_{pruned} \in \mathcal{RT}$ a subset of rooted trees which contains only pruned rooted trees obtained by application of this procedure which we denote by the function $Prune : \mathcal{RT} \to \mathcal{RT}_{pruned}$. Consider each tree to be pruned. After performing one of the operations $\cup$, $\widehat{\ }$, or $\|$ the new tree may no longer be pruned, therefore we need to perform the pruning of the new tree every time.

The height function h defined earlier is applied to pruned trees and has one special case for the tree with only one edge labeled with τ; for these trees (with a white or black node) it returns the height 0.

Proposition 2.4 (characterization of pruned trees [PS07a])**.**

1. *Any pruned tree either contains no black nodes, or it is the tree with only one edge* $(r, \tau, n : B)$ *labeled with* τ *and ending in a black node.*

2. *A pruned tree has no label* α *which contains* τ *unless* $\alpha = \tau$ *and it labels an edge* $(n, \tau, m{:}W)$ *and* m *is a leaf node with siblings or it is the tree with only one edge* (r, τ, n)*.*

The size of a tree is the number of nodes together with the number of edges; i.e. $size(T) = |\mathcal{N}| + |E|$.

Proposition 2.5 (time complexities)**.**

1. *The time complexity for the* join *and* concatenation *operations over trees is linear in the size of the trees.*

2. *The time complexity for the* concurrent join *operation of two trees is polinomial in the size of the two trees.*

3. *The* pruning *procedure is linear in the size of the tree that is pruned.*

From the proof above it is easy to see the following:

1. The size of the tree resulting from the *join* or *concatenation* is at most the sum of the sizes of the two trees.

2. The size of the tree resulting from the *concurrent join* is at most polinomial in the sizes of the two trees.

3. The size of a *pruned* tree is clearly at most the size of the initial tree.

2.2.2 Interpreting the actions

In this section we give a standard interpretation of the elements of the algebra $\mathcal{CA}$ and of the algebraic operators using the rooted trees and the operations defined in Section 2.2.1. For this we construct a map $I_{\mathcal{CA}}$ which maps every action of $\mathcal{CA}$ into a rooted tree and preserves the structure imposed by the constructors. This means that $I_{\mathcal{CA}}$ is the homomorphic extension of $\bar{I}_{\mathcal{CA}} : \mathcal{A}_B \cup \{\mathbf{0}, \mathbf{1}\} \to \mathcal{RT}$ which is the map over the constructors (i.e. the basic actions and the two special actions) of $\mathcal{CA}$.

1. The definition of $\bar{I}_{\mathcal{CA}}(a)$ for basic actions $a \in \mathcal{A}_B$ returns a basic rooted tree $T_a = (\{r, n\}, \{(r, a, n)\}, \{a\})$ with only one edge labeled with a and with $n : W$ a white node.

2. For the special actions $\mathbf{1}$ and $\mathbf{0}$ we have respectively the trees:

   (a) $\bar{I}_{\mathcal{CA}}(\mathbf{1}) = (\{r, n\}, \{(r, \tau, n)\}, \{\tau\})$ with $n : W$

   (b) $\bar{I}_{\mathcal{CA}}(\mathbf{0}) = (\{r, n\}, \{(r, \tau, n)\}, \{\tau\})$ with $n : B$

Informally the skip action $\mathbf{1}$ means not performing any action and its interpretation as an edge with an empty set of labels goes well with the intuition. The fail action $\mathbf{0}$ is interpreted as taking the path into a black node.

We now extend $\bar{I}_{\mathcal{CA}}$ from basic actions to compound actions of $\mathcal{A}$ using an inductive definition, and obtain a homomorphism $I_{\mathcal{CA}} : \mathcal{CA} \to \mathcal{RT}$.

3. $I_{\mathcal{CA}}(\alpha + \beta) = I_{\mathcal{CA}}(\alpha) \cup I_{\mathcal{CA}}(\beta)$;

4. $I_{\mathcal{CA}}(\alpha \cdot \beta) = I_{\mathcal{CA}}(\alpha) \widehat{} I_{\mathcal{CA}}(\beta)$;

5. $I_{\mathcal{CA}}(\alpha \& \beta) = I_{\mathcal{CA}}(\alpha) \parallel I_{\mathcal{CA}}(\beta)$.

We still need to take care of the conflict relation $\#_{\mathcal{C}}$ of the algebra with respect to the concurrency operator &; i.e. we need to interpret the fact that $a \& b = \mathbf{0}$ if $a \#_{\mathcal{C}} b$. It is easy to define the same compatibility relation over the basic actions of the algebra for the labels of the rooted trees. With this definition we enforce each label of an edge of the form $(m, \{\alpha, \beta\}, n)$ with $\alpha \#_{\mathcal{C}} \beta$ and $n : W$ to be replaced by the τ label and $n : B$ becomes a black node.

Note that the length of an action of $\mathcal{CA}$ corresponds to the height of the interpretation of the action as a rooted tree. Because we always prune the trees (and work only with pruned trees) we consider the function $\hat{I}_{\mathcal{CA}} : \mathcal{CA} \to \mathcal{RT}_{pruned}$ which is defined as $\hat{I}_{\mathcal{CA}} = Prune \circ I_{\mathcal{CA}}$. Note that $\hat{I}_{\mathcal{CA}}$ is

not a homomorphism and can be proven by giving a counter example to the requirement $\hat{I}_{\mathcal{CA}}(\alpha + \beta) = \hat{I}_{\mathcal{CA}}(\alpha) \cup \hat{I}_{\mathcal{CA}}(\beta)$.[12] This means that the function *Prune* is not homomorphic which means that after composing (using any of the three operations $\cup, \widehat{\ }, \|$) two pruned rooted trees the function *Prune* has to be applied again. On the other hand Lemmas 2.6, 2.7, and 2.11 give other useful properties of the *Prune* function.

Lemma 2.6 ([PS07a])**.** *If* $Prune(T_1) = Prune(T_2)$ *and* T_1' *and* T_2' *are subtrees of respectively* T_1 *and* T_2 *s.t. there is the same path from* r_{T_1} *to* $r_{T_1'}$ *and from* r_{T_2} *to* $r_{T_2'}$ *which contains no black node, then* $Prune(T_1') = Prune(T_2')$.

Lemma 2.7 ([PS07a])**.** *The function Prune preserves the substitution property of the equality* $=$ *on guarded trees.*

Take $[op] \in \{\cup, \widehat{\ }, \|\}$ *then*
if $Prune(T_1) = Prune(T_1')$ *and* $Prune(T_2) = Prune(T_2')$ *then*
$Prune(T_1[op]T_2) = Prune(T_1'[op]T_2')$.

Lemma 2.7 suggests the following result.

Corollary 2.8. $\forall \alpha, \alpha', \beta, \beta' \in \mathcal{CA}$ *if* $\hat{I}_{\mathcal{CA}}(\alpha) = \hat{I}_{\mathcal{CA}}(\alpha')$ *and* $\hat{I}_{\mathcal{CA}}(\beta) = \hat{I}_{\mathcal{CA}}(\beta')$ *then* $\hat{I}_{\mathcal{CA}}(\alpha[op]\beta) = \hat{I}_{\mathcal{CA}}(\alpha'[op]\beta')$ *where* $[op] \in \{+, \cdot, \&\}$.

Theorem 2.9 (Completeness of $\mathcal{CA}$ over $\mathcal{RT}$ [PS07a])**.**

For any two actions α *and* β *of* $\mathcal{A}$ *then* $\alpha = \beta$ *is a theorem of* $\mathcal{CA}$ *iff the corresponding trees* $\hat{I}_{\mathcal{CA}}(\alpha)$ *and* $\hat{I}_{\mathcal{CA}}(\beta)$ *are isomorphic (i.e. equal renaming of the nodes).*

Note: logicians would call the forward implication the *soundness* and the backword implication the *completeness*.

Proof: The forward implication ($\Rightarrow$) can be rewritten as:

$$\mathcal{CA} \vdash \alpha = \beta \Rightarrow \hat{I}_{\mathcal{CA}}(\alpha) \doteq \hat{I}_{\mathcal{CA}}(\beta)$$

The relation $\vdash$ is the classical derivation relation from equational logic. It meas that $\alpha = \beta$ can be derived from the axioms of the $\mathcal{CA}$ algebra using the derivation rules of reflexivity, symmetry, transitivity, and substitution. We use induction on the derivation and prove as base case that the implication holds for the axioms of $\mathcal{CA}$. The rooted trees in the theorem are only pruned trees. Thus, after the standard interpretation generates a tree, then the tree is pruned. The inductive step considers the derivation rules.

For the converse implication ($\Leftarrow$) of the theorem we need to prove that the standard interpretation restricted to pruned tress $\hat{I}_{\mathcal{CA}}$ is an isomorphism

[12] The counterexample is $\hat{I}_{\mathcal{CA}}(a + \mathbf{0}) \neq \hat{I}_{\mathcal{CA}}(a) \cup \hat{I}_{\mathcal{CA}}(\mathbf{0})$.

up to the axioms of the $\mathcal{CA}$ algebra. This means that if $I_{\mathcal{CA}}(\alpha)$ is applied to action α it returns a normal rooted tree T_α which is then pruned and from the pruned tree one can get by applying an inverse function another action α'. The obtained action α' has to be equal by the axiom system of Table 3 with the original action $\alpha = \alpha'$. Having the isomorphism up to, then from two actions α and β we get the same tree T_γ from where we translate back using the inverse mapping, to the same action $\gamma = \alpha = \beta$ which is our conclusion.

First we take the usual way of defining a relation induced by the equality on action terms and the derivation relation $\vdash$.

Definition 2.10. *Consider the relation* $\equiv \subseteq T_{\mathcal{CA}} \times T_{\mathcal{CA}}$ *defined as:*

$$\alpha \equiv \beta \Leftrightarrow \mathcal{CA} \vdash \alpha = \beta$$

The proof that $\equiv$ is a congruence is classical and uses the deduction rules and we leave it to the reader.

The rest of the proof is based on the following lemma which essentialy establishes the existence of the inverse function of the standard interpretation, thus proving that $\hat{I}_{\mathcal{CA}}$ is an isomorphism up to $\equiv$.

Lemma 2.10 (Existence of the inverse of the interpretation)**.**

There exists a map $\hat{I}^{-1}_{\mathcal{CA}} : \mathcal{RT}_{pruned} \to \mathcal{CA}$ *which is the inverse map up to* $\equiv$ *of* $\hat{I}_{\mathcal{CA}}$.

Proof: The proof of the lemma involves three parts:

1. $\forall \hat{T} \in \mathcal{RT}_{pruned}$ then $\exists \alpha \in \mathcal{CA}$ s.t. $\hat{I}^{-1}_{\mathcal{CA}}(\hat{T}) = \alpha$.
2. $\forall \hat{T}_1 \doteq \hat{T}_2$ then $\hat{I}^{-1}_{\mathcal{CA}}(\hat{T}_1) = \hat{I}^{-1}_{\mathcal{CA}}(\hat{T}_2)$.

   The first two guarantee that $\hat{I}^{-1}_{\mathcal{CA}}$ is a correctly defined function and their proof will be part of the construction of $\hat{I}^{-1}_{\mathcal{CA}}$.
3. $\hat{I}^{-1}_{\mathcal{CA}} \circ \hat{I}_{\mathcal{CA}} = Id/_{\equiv}$ i.e. $\forall \alpha \in \mathcal{CA}$ then $\hat{I}^{-1}_{\mathcal{CA}} \circ \hat{I}_{\mathcal{CA}}(\alpha) = \alpha'$ and $\alpha \equiv \alpha'$.

We define $\hat{I}^{-1}_{\mathcal{CA}}$ as the restriction of the function $I^{-1}_{\mathcal{CA}} : \mathcal{RT} \to \mathcal{CA}$ to $\mathcal{RT}_{pruned}$ the set of pruned trees. Note that one should not regard the notation $I^{-1}_{\mathcal{CA}}$ as the inverse function of $I_{\mathcal{CA}}$, our intension is just to keep an intuitive notation. The construction of $I^{-1}_{\mathcal{CA}}$ is first done for the basic trees and in the second stage it is extended homomorphic to the tree operators. The set of basic trees (nontrivial ones) contains the trees with only one edge labeled with a basic action or τ; i.e. $\{T_B = (\{r, n\}, \{(r, \delta, n)\}, \{\delta\}) \mid \delta \in \mathcal{A}_B \cup \{\tau\}$ and $n : W$ or $n : B\}$. The *Prune* function transforms all basic

trees with black nodes and a label $a \neq \tau$ into a basic tree with label τ. This means that $\hat{I}^{-1}_{\mathcal{CA}}$ is applied only to trees with labels $a \neq \tau$ and white nodes for which it returns the action $a \in \mathcal{A}_B$, the tree labeled with τ and white node for which it returns action $\mathbf{1}$, and to the tree labeled with τ and black node for which it returns action $\mathbf{0}$.

The extension of $I^{-1}_{\mathcal{CA}}$ to the tree operators is natural:

- $I^{-1}_{\mathcal{CA}}(T_1 \cup T_2) = I^{-1}_{\mathcal{CA}}(T_1) + I^{-1}_{\mathcal{CA}}(T_2)$
- $I^{-1}_{\mathcal{CA}}(T_1 \widehat{\ } T_2) = I^{-1}_{\mathcal{CA}}(T_1) \cdot I^{-1}_{\mathcal{CA}}(T_2)$
- $I^{-1}_{\mathcal{CA}}(T_1 \parallel T_2) = I^{-1}_{\mathcal{CA}}(T_1) \& I^{-1}_{\mathcal{CA}}(T_2)$

With this construction we have proven that $I^{-1}_{\mathcal{CA}}$ is defined on the whole domain $\mathcal{RT}$ and thus $\hat{I}^{-1}_{\mathcal{CA}}$ is defined on the whole $\mathcal{RT}_{pruned}$. Now we have to prove that it returns a unique value for each input, in order to call it a function.

Note that the definition of $\hat{I}^{-1}_{\mathcal{CA}}$ does not take into consideration the names of the nodes of the trees thus, for any two trees $\hat{T}_1 \doteq \hat{T}_2$ it will return the same action. It remains to show that if two trees are equal in the usual sense ($\hat{T}_1 = \hat{T}_2$) then the function $\hat{I}^{-1}_{\mathcal{CA}}$ returns the same action. This is obvious as two equal trees have the same nodes, the same edges (with the same labels), and thus the same structure. It does not matter the order of the edges of a node or the order of the basic labels in a compound label, but these are dealt with at the level of the actions by the commutativity of the + and the commutativity of the & operators. Consider just the following case when two branches of a tree are interchanged so to give a second tree. This gives the same action in the algebra modulo commutativity axiom. We conclude that $\hat{I}^{-1}_{\mathcal{CA}}$ is a well defined function.

Lemma 2.11. *Function Prune preserves the relation $\equiv$ on actions, meaning that $\forall T \in \mathcal{RT}$ if $Prune(T) = \hat{T}$ then $I^{-1}_{\mathcal{CA}}(T) \equiv I^{-1}_{\mathcal{CA}}(\hat{T})$.*

From Lemma 2.11 we conclude the following useful congruences:

$$I^{-1}_{\mathcal{CA}} \circ I_{\mathcal{CA}} \equiv I^{-1}_{\mathcal{CA}} \circ \hat{I}_{\mathcal{CA}} \tag{22}$$

which by restriction implies

$$I^{-1}_{\mathcal{CA}} \circ I_{\mathcal{CA}} \equiv \hat{I}^{-1}_{\mathcal{CA}} \circ \hat{I}_{\mathcal{CA}} \tag{23}$$

The proof of part 3 of Lemma 2.10 uses structural induction on the structure of the action α. Proving part 3 we prove that $\hat{I}^{-1}_{\mathcal{CA}}$ is the isomorphic image of $\hat{I}_{\mathcal{CA}}$ up to the congruence on actions $\equiv$.

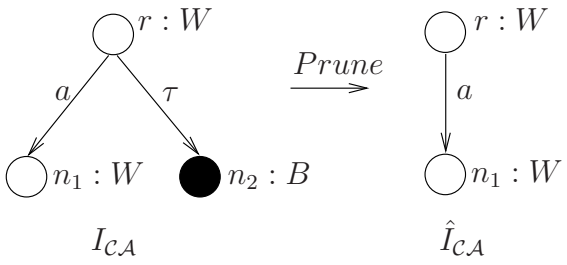

Figure 5: Example of applying the isomorphism $\hat{I}_{\mathcal{CA}}$.

Consider as an example the special case when $\alpha = a + \mathbf{0}$ which is pictured in Figure 5. $I_{\mathcal{CA}}(\alpha) = (\{r, n_1, n_2\}, \{(r, a, n_1), (r, \tau, n_2)\}, \{a, \tau\})$ with $n_1 : W$ and $n_2 : B$. Applying the *Prune* function we obtain the tree $\hat{T}_\alpha = (\{r, n_1\}, \{(r, a, n_1)\}, \{a\})$, where applying the $\hat{I}^{-1}_{\mathcal{CA}}$ we obtain the action $a \in \mathcal{CA}$. We have that $\mathcal{CA} \vdash a + \mathbf{0} = a$ as an instance of the axiom (3) of Table 3 and thus we have our conclusion $\alpha = a + \mathbf{0} \equiv a = \hat{I}^{-1}_{\mathcal{CA}} \circ \hat{I}_{\mathcal{CA}}(\alpha)$. □

To finish the proof of the second implication, i.e. $\hat{I}_{\mathcal{CA}}(\alpha) \doteq \hat{I}_{\mathcal{CA}}(\beta) \Rightarrow \mathcal{CA} \vdash \alpha = \beta$ we make use of Lemma 2.10. From $\hat{I}_{\mathcal{CA}}(\alpha) \doteq \hat{I}_{\mathcal{CA}}(\beta)$ we apply $\hat{I}^{-1}_{\mathcal{CA}}$ and obtain $\alpha' = \hat{I}^{-1}_{\mathcal{CA}} \circ \hat{I}_{\mathcal{CA}}(\alpha)$ and $\beta' = \hat{I}^{-1}_{\mathcal{CA}} \circ \hat{I}_{\mathcal{CA}}(\beta)$ with $\alpha' = \beta'$ as hypothesis and $\alpha \equiv \alpha'$ and $\beta \equiv \beta'$ from Lemma 2.10. Thus we have the conclusion $\alpha \equiv \alpha' = \beta' \equiv \beta$ which is $\mathcal{CA} \vdash \alpha = \beta$. □

We can take another way of viewing the rooted trees as the set of all paths starting from the root node. This is similar to the way of giving semantics to actions in process logic [Pra79] where each action is interpreted as a set of trajectories.

2.3 The Boolean tests

In this section we extend $\mathcal{CA}$ with a Boolean algebra of tests to obtain an action algebra with tests which we denote by $\mathcal{CAT}$; we follow the work of Kozen [Koz97] on defining Kleene algebra with tests.

The structure $\mathcal{CAT} = (\mathcal{CA}, \mathcal{B})$ combines the previous defined algebraic structure $\mathcal{CA}$ with a Boolean algebra $\mathcal{B}$ in a special way we see in this section. A Boolean algebra is a structure $\mathcal{B} = (\mathcal{A}_\mathbf{1}, \vee, \wedge, \neg, \bot, \top)$ where the function symbols ($\vee$, $\wedge$, and $\neg$) and the constants ($\bot$ and $\top$) have the usual meaning of disjunction, conjunction, negation, falsity, and truth respectively. Moreover, the elements of set $\mathcal{A}_\mathbf{1}$ are called *tests* and are included in the set of actions of the $\mathcal{CA}$ algebra (i.e. tests are special actions; $\mathcal{A}_\mathbf{1} \subseteq \mathcal{A}$). We denote tests by letters from the end of the Greek alphabet $\phi, \varphi, \ldots$ followed by a

question mark ?. Our notation for tests is more related to the notation used in Propositional Dynamic Logic (PDL).

For a more clear presentation, we abuse the syntax and use, e.g. $(\phi \wedge \varphi)?$ instead of $\phi? \wedge \varphi?$. More generally, we consider ? only at the end of an expression from $\mathcal{A}_1$; i.e. if ψ is a test expression generated using any combination of the constructors of the boolean algebra then the notation for the test is just $\psi?$.

The intuition behind tests is that in an action $\phi? \cdot \alpha$ formed of a test $\phi?$ followed by an action α is the case that action α can be performed only if the test succeeds (the condition ϕ is satisfied). Tests are sometimes called guards and have been used to model *while programs* which involve programming constructs like loops and conditionals. For example, consider the `if` ϕ `then` a `else` b programming construct. We can model this using tests as: $\phi? \cdot a \ + \ \neg\phi? \cdot b$. Some other properties of systems could be modelled by giving equations involving both actions and tests. For example the following commutative equation $\phi? \cdot \alpha = \alpha \cdot \phi?$ models an *action invariant*[13]; i.e. if ϕ is true before action α then we should consider it also true after performing α.

We do not go into details about the properties of a Boolean algebra as these are classical results in the literature. For a more thorough understanding see [Koz97] and references therein. In the reminder of this section we present the relation between tests and actions.

The first relation between the $\mathcal{CA}$ algebra and the boolean algebra $\mathcal{B}$ is that $\top? = \mathbf{1}$ with the intuition that testing a tautology always succeeds. The dual is $\bot? = \mathbf{0}$ meaning that testing a falsity never succeeds. Furthermore, $\mathbf{1} \cdot \alpha = \alpha = \top? \cdot \alpha$ which is obvious as testing a tautology always succeeds so the action α can always be performed. For the dual we have $\mathbf{0} \cdot \alpha = \mathbf{0} = \bot? \cdot \alpha$ with the intuition that because testing a falsity never succeeds the action α is never performed (the sequence of actions stops when it reaches the falsity test).

We consider the sequence actions as strings separated by $\cdot$ constructor. With the extension with tests we no longer have strings but *guarded strings* [Kap69]. A guarded string (in our algebra) is a sequence of actions interplaced with tests; e.g. $\phi_1? \ a \ \phi_2? \ \phi_3? \ b \ \phi_4?$ is a guarded string (recall that we sometimes omit the $\cdot$ for brevity). Moreover, note that is not necessary to have more tests in a row because the sequence of test actions from $\mathcal{CA}$ algebra is pushed inside the boolean algebra and shrunken into only one test with the use of conjunction operator of $\mathcal{B}$; e.g. $\phi_2? \ \phi_3? = (\phi_2 \wedge \phi_3)?$. Thus a

[13]Performing any action α does not affect the truth value of proposition ϕ.

guarded string is an alternation of tests and actions:

$$\phi_0? \; \alpha_1 \; \phi_1? \ldots \alpha_n \; \phi_n? \tag{24}$$

Note that a normal string of actions is a guarded string where instead of tests $\phi_i?$ we have the tautology test $\top? = \mathbf{1}$.

For a better intuition of tests and actions we give the following example. Take the test $\phi?$ to be: "The bank account is less than 500$", and an action a of "deposit 1000$". We can give the more complex action $\phi? \cdot a$ which states that "when the bank account is less than 500$ deposit 1000$ into the account". Another example is: $\phi?$ to test that "The bank account is less than 500$" and $\varphi?$ to test that "The bank account is greater than 500$". The complex test $(\phi \wedge \varphi)?$ which (because $\varphi = \neg\phi$) is an instance of the general falsity test $(\phi \wedge \neg\phi)?$ never succeeds. The example is that whenever one waits to "deposit 1000$" after the test (of falsity) $(\phi \wedge \varphi)?$ succeeds then the action of depositing the money will never be performed.

We extend the definition of the length function to apply it also to tests. As we have seen the relation between $\mathbf{1}$ and $\top?$ we consider the length of a test is 0; i.e. $l(\phi?) = 0$. In other words, the length function does not take into consideration the tests; e.g. $l(\phi? \; a) = l(\phi? \; a \; \varphi?) = 1$. Moreover, the position syntax $\alpha(n)$ skips the tests; e.g. if $\alpha = \phi_1? \; a \; \phi_2? \; \phi_3? \; b \; \phi_4?$ then $\alpha(2) = b$.

Other relations between $\mathcal{CA}$ and $\mathcal{B}$ are:

1. concurrent composition of two tests $\phi?\&\varphi?$ is $(\phi \wedge \varphi)?$ which is the same as the sequence of two tests.

2. concurrent composition of a test and an action $\phi?\&a$ or $a\&\phi?$ is the same as the sequence of the test and the action $\phi? \cdot a$. An intuitive motivation for this is that when performing at the same time an action and a test one expects that the test is satisfied before the completion of the action; and because we do not have a notion of start and end of an action we have to consider the test before the action. This approach is also motivated by the fact that an action can change the world and thus the test may hold no longer.

3. concurrent composition of two sequence actions each formed of one test followed by an actions; i.e. $(\phi? \cdot a)\&(\varphi? \cdot b)$ which is $(\phi \wedge \varphi)? \cdot a\&b$. Note that this way of concurrently composing sequence of tests and actions conforms with the axiom (17) of $\mathcal{CA}$. More precisely, recall that the length function (and the position syntax) for guarded strings of actions do not take into consideration the tests. Thus, in the general

concurrent composition of two guarded strings $\phi_0?\ \alpha_1\ \phi_1? \ldots \alpha_n\ \phi_n?$ and $\varphi_0?\ \beta_1\ \varphi_1? \ldots \beta_m\ \phi_m?$ the axiom considers pairs of $(\phi_0? \cdot \alpha_1) \& (\varphi_0? \cdot \beta_1)$.

Example 2.1. *Let us consider some simple examples.*

i. The action $a\&(\varphi? \cdot b)$ is an instance of the case 3. above where action a is preceded by test $\top?$. Thus, the action is the same as $(\top \wedge \varphi)? \cdot (a\&b) = \varphi? \cdot (a\&b)$.

ii. An example of the distributivity axion (9) of Table 3 is the action $(a + \phi?) \cdot b$ which is equivalent to $a \cdot b + \phi? \cdot b$.

iii. The action $(a\&\phi?) \cdot b$ transforms using the case 2. above into $\phi? \cdot a \cdot b$.

The syntactic structure of the actions in $\mathcal{CAT}$ is given through defining the term algebra $T_{\mathcal{CAT}}$. In the following we define inductively two kinds of terms: the *boolean terms* and the *action terms*. The carrier set of the term algebra $T_{\mathcal{CAT}}$ is the set of all action terms.

Definition 2.11 (action terms of $\mathcal{CAT}$)**.**

1. *$\top?$ and $\bot?$ are boolean terms;*
2. *if $\phi?$ and $\varphi?$ are boolean terms then $(\phi \wedge \varphi)?$, $(\phi \vee \varphi)?$, and $\neg\phi?$ are boolean terms;*
3. *nothing else is a boolean term;*
4. *any boolean term is an action term;*
5. *any basic action $a \in \mathcal{A}_B$ is an action term;*
6. *if α and β are action terms then $\alpha\&\beta$, $\alpha \cdot \beta$, and $\alpha + \beta$ are action terms;*
7. *nothing else is an action term.*

2.4 Standard interpretation of $\mathcal{CAT}$ over guarded rooted trees

In this section we extend the rooted trees of Section 2.2.1 with tests and call them *guarded rooted trees*. On this trees we give the standard interpretation of the $\mathcal{CAT}$ algebra.

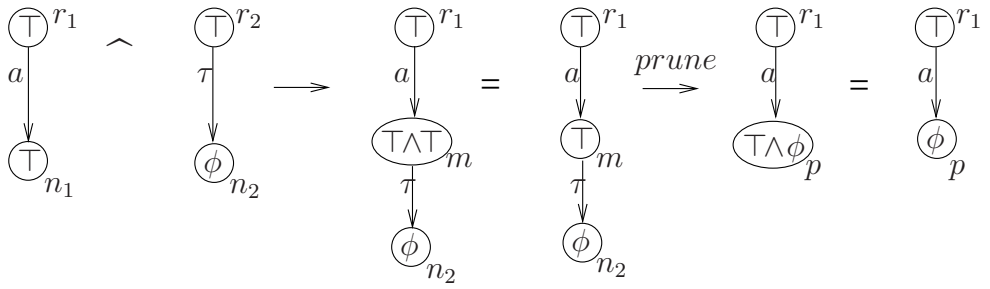

Figure 6: Example of concatenation of two guarded rooted trees.

The extension is simple by associating with each node a boolean expression ϕ. We denote the new nodes by $n : \{\phi\}$ where ϕ is generated by the Boolean algebra $\mathcal{B}$ of Section 2.3. The two colors of the nodes are now special cases in the extended trees: a white node is $n : \{\top\}$ and the black node is $n : \{\bot\}$.

All the constructions for rooted trees are the same with minor modifications. The operators $\cup$, $\widehat{\ }$, and $\|$ for guarded rooted trees when combining two nodes $n_1 : \{\phi\}$ and $n_2 : \{\varphi\}$ make the conjunction of the Boolean expressions into $n_{12} : \{\phi \wedge \varphi\}$. The pruning procedure also adheres to this conjunction of the Boolean expressions of the two nodes that need to be combined.

Example 2.2. *We give in Figure 6 an example of concatenating two trees; the first representing an action a and the second consisting of an empty label τ and a test ϕ. The resulting guarded rooted tree represents the action of performing a after which the Boolean expression ϕ is tested. Note that in a first step the two trees are just combined using the concatenation operation and only afterwards the tree is pruned by removing the τ edge. In the first step the combination of the nodes n_1 and r_2 into m gives the expression $\top \wedge \top = \top$. After pruning of the tree the nodes m and n_2 are combined into p with the resulting conjunction $\top \wedge \phi$ which is the same as just ϕ.*

The standard interpretation of $\mathcal{CAT}$ algebra over the guarded rooted trees is given through a map $I_{\mathcal{CAT}}$ which maps every action term of $T_{\mathcal{CAT}}$ into a guarded rooted tree and preserves the structure imposed by the constructors. $I_{\mathcal{CAT}}$ is the same as $I_{\mathcal{CA}}$ of Section 2.2 with the following differences:

1. for basic actions $a \in \mathcal{A}_B$ the white nodes of the trees are replaced by nodes with the $\top$ expression inside; i.e. $r : \{\top\}$ and $n : \{\top\}$.

2. the special actions $\mathbf{1}$ and $\mathbf{0}$ have the white nodes replaced with $\top$ and the black node with $\bot$; i.e. $n_{\mathbf{1}} : \{\top\}$ and $n_{\mathbf{0}} : \{\bot\}$.

3. the tree operators are changed as discussed above.

4. the major difference is that $I_{\mathcal{CAT}}$ interprets test ϕ.

   $I_{\mathcal{CAT}}(\phi) = (\{r, n\}, \{(r, \tau, n)\}, \{\tau\})$ and $n : \{\phi\}$ is the tree with one edge labeled with τ and the leaf node has ϕ inside.

As we have discussed there is only one test needed between any two actions in the $\mathcal{CAT}$ algebra. At the level of the guarded trees this is respected because of the pruning and of the conjunction of the expressions inside the combined nodes. The interpretation of tests gives trees with hight 0 (as we have seen for **1** and **0** earlier). Note that we can still interpret an action formed of only one test.

We conjecture here that the algebra $\mathcal{CAT}$ is complete with respect to the guarded rooted trees, and the proof is similar to the proof of the corresponding Theorem 2.9 for $\mathcal{CA}$ algebra.

2.5 Canonic Form of Actions

One of the purposes of the investigation of the algebra in this report is to be able to give a natural notion of *action negation*. There have been a few works related to negation of actions [Mey88, HTK00, LW04, Bro03]. In [Mey88], the same as in [HTK00] action negation is with respect to the universal relation which, for example 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 basically the complement of an action (not with respect to the universal relation but) with respect to 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.

It is known that for regular expressions there is no standard normal form; for example, see the *Starr-Height problem* [Egg63] which looks at regular expressions normal forms from the perspective of Kleene *. Similarly, there is no action normal form for the action algebra of PDL.

A first attempt to identify a normal form for the classical action operators of Kleene algebra *choice* $\cup$, *sequence* ;, and *Kleene star* * underlying PDL is:

$$\alpha = \bigcup_{a \in A} a; \alpha'$$

where α is a compound action, a is an atomic action, A is a subset of atomic actions, and α' is in normal form. For the semantics of actions given with trajectories, as in Process Logics [Pra79] this way of representing actions gives all the trajectories of an action.

The problem with this definition is that it takes into account the * operator which has an infinitary interpretation as the reflexive transitive closure on binary relations. Looking at its unfolding $a^* = \mathbf{1} + a + a \cdot a + \ldots$ it respects the normal form above. But, when we take one of the equations that define it; $a^* = \mathbf{1} + a \cdot a^*$ it is clear that we can not prove the existence of the normal form. This is because the normal form of $\alpha = a^*$ would be based on the fact that $\alpha' = a^*$ is in normal form, and we get nontermination.

For our action algebra $\mathcal{CAT}$ defined in Section 2 we have a canonic form similar to the one above. The definition below shows how any action term constructed by the term algebra $T_{\mathcal{CAT}}$ can be written in a concise and clear way.

Definition 2.12 (canonic form for $\mathcal{CAT}$)**.** *For any restricted action α defined with the operators $+$, $\cdot$, $\&$, and $?$ it exists a canonic form denoted by ACF^α and defined as follows, where $\alpha^i_\& \in \mathcal{A}^\&_B$ or $\alpha^i_\&$ is a test and α^i is a compound action in canonic form.*

$$\alpha = \underset{i \in I}{+} \alpha^i_\& \cdot \alpha^i$$

The indexing set I is finite as the compound actions α are finite; i.e. there is a finite number of application of the $+$ operator.

Theorem 2.12 ([PS07a])**.** *For every action α of the algebra $\mathcal{CAT}$ we have a corresponding ACF^α.*

2.6 Action negation as a derived operator

A natural and useful view of *action negation* is to say that the negation $\overline{\alpha}$ of action α is the action given by all the immediate trajectories that *take us outside* the trajectory of α [BWM01]. With ACF^α it is easy to formally define $\overline{\alpha}$.

Definition 2.13 (action negation)**.** *The action negation is denoted by $\overline{\alpha}$ and is defined as:*

$$\overline{\alpha} = \overline{\underset{i \in I}{+} \alpha^i_\& \cdot \alpha^i} = \underset{\gamma \in \overline{R}}{+} \gamma \; + \; \underset{i \in I}{+} \alpha^i_\& \cdot \overline{\alpha^i}$$

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

$$\overline{R} = \{(\neg\phi)? \,|\, \phi \in R\} \cup \{\gamma \,|\, \gamma \in \mathcal{A}^\&_B, \text{ and } \forall i \in I, \alpha^i_\& \not\leq_\& \gamma\}.$$

We include *action negation* as a derived operator of the $\mathcal{CAT}$ algebra. Action negation may be considered as a function $\overline{\ }: \mathcal{A} \to \mathcal{A}$ which takes as argument a compound action in canonic form and returns another compound action (i.e. action negation is not a constructor of the algebra).

In the construction of the set $\overline{R}$ we include several things. First we look at the negation of a single test $\overline{\phi?}$ (i.e. when $\alpha^i_{\&}$ is the test $\phi?$) which is just the negated test in the Boolean algebra $(\neg\phi)?$. Therefore, the negation of *skip* $\overline{\mathbf{1}} = \overline{\top?} = \bot? = \mathbf{0}$ is the violation action and vice-cersa $\overline{\mathbf{0}} = \mathbf{1}$.

Let us take one particular case of the negation of the choice between two basic actions $\overline{a+b}$. Any action which does not "contain" neither a nor b is part of the negation of $a+b$; e.g. $a\&c \notin \overline{a+b}$, but $c, c\&d \in \overline{a+b}$. Formally, any concurrent action $\beta \in \mathcal{A}^{\&}_B$ with the properties that $a \not\leq_{\&} \beta$ and $b \not\leq_{\&} \beta$ is a negation of $a+b$. A more clear understanding of action negation will come after reading the proof of Proposition 3.11 which gives a characterization of action negation in terms of traces of actions.

The negation of a action says intuitively that the action which is negated is not executed. This is why we do not include in the negation of an action (eg. a) a concurrent action which is more demanding than a, i.e. contains a (eg. executing action $a\&b$ means that action a is executed, so it cannot be included in the negation of a).

From *Example 1.* consider the negation $\overline{p + d\&ne}$ of the action "pay or delay and notify by e-mail". Any action γ which does not "contain" neither p nor $d\&ne$ (i.e. $p \not\leq_{\&} \gamma$ and $d\&ne \not\leq_{\&} \gamma$) is part of the negation of $p + d\&ne$; e.g. $p\&ebl \notin \overline{p + d\&ne}$, but $d, ebl, d\&ebl \in \overline{p + d\&ne}$.

Discussion: 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.

We consider *active systems* which are systems that always do an action. It is simple to model passivity by action $\mathbf{1}$ *skip*. If we add time for actions (i.e. the duration of an action) and with time it is natural to model idling by the *skip* action with a certain duration. Thus, not doing action a may be represented by doing action *skip* or may be represented by doing another explicit action. When adding time the "doing" of an action will become more complicated.

Proposition 2.13 (action negation is an action)**.**
The negation operator returns an action.

Proof: We have to prove that the action returned by the negation operation $\overline{\ }$ is indeed an action of $\mathcal{CAT}$. This means that we need to prove that there is no infinite branching and also no infinite depth in the action negation. This is fairly simple to see.

As the actions α are finite then they have finite branching, thus the indexing set I is finite. Therefore the only possible source of infinite branching is $\overline{R}$. On the other hand $\overline{R}$ is a finite set because it contains elements of $\mathcal{A}_B^{\&}$ which is finite and for each test $\phi?$ it contains a test $\neg\phi?$ thus a finite number of tests also.

It remains to argue that $\overline{\alpha}$ has finite depth. The first part of the action negation (i.e. $+_{\gamma\in\overline{R}}\gamma$) introduces only finite branches. Actually this is the part which ends each branch of the action negation. Therefore the only source of infinite depth may be the second part (i.e. $+_{i\in I}\ \alpha_{\&}^i \cdot \overline{\alpha^i}$). Note that the definition of action negation is a recursive one and also note that it goes stepwize; meaning at each recursion step it decreases the length of the action it applies to. Naturally at some point (because the action α has finite depth) α^i will become $\mathbf{1}$. At this point the recursive application of $\overline{\ }$ stops in a $\mathbf{0}$ action. □

Note: The negation of an action α from Definition 2.13 is in canonic form. It respects the canonic form because it is a choice between sequences of a action (i.e. γ or $\alpha_{\&}^i$) which is only a basic actio, a concurrent action or a test; and each of these are followed by a action which is also in canonic form (i.e. respectively $\mathbf{1}$ and $\overline{\alpha^i}$ which is in canonic form by structural induciton).

Definition 2.14 (representation independence)**.** *Consider a set S equiped with an equivalence relation $=$. We call a function $f : S \to S$* ***independent of the representation*** *iff $\forall a, b \in S$ then if $a = b$ implies $f(a) = f(b)$.*

Intuitively, if a function is independent of the representation (of the elements in its domain) it means that it takes all elements from one equivalence class to the same (possible different) equivalence class. In other words the result of the application of the function on any of the elements of an equivalence class is *the same* (i.e. is in the same equivalence class) as the result of the application of f on the representant element of the equivalence class.

Consider the dynamic box modality of $\mathcal{CL}$ as a function $[\ ]\mathcal{C} : \mathcal{CAT} \to \mathcal{CL}$ thich takes as argument an action of the $\mathcal{CAT}$ algebra (N.B. not an action of PDL; i.e. not having the Kleene * operator) and returns a formula in the $\mathcal{CL}$ logic. It is proven in Proposition 3.10 that the box modality is indeed independent of the representation of the actions. On the other hand the next result is a negative one as it shows that the negation operator $\overline{\ }$ is *not* independent of the representation.

Proposition 2.14. *The function* $\overline{\ }: \mathcal{A} \to \mathcal{A}$ *is not independent of the representation of the actions of* $\mathcal{A}$.

Proof: The proof is immediate by considering a counterexample. Take the actions a and $a + b \cdot \mathbf{0}$ where $\mathcal{A}_B = \{a, b, c\}$ contains three basic actions. Obviously $a = a + b \cdot \mathbf{0}$ by axioms (7) and (3). Then by Definition 2.13 we have that $\overline{a} = c + b + b\&c + a \cdot \mathbf{0} \overset{(7),(3)}{=} c + b + b\&c$ and that $\overline{a + b \cdot \mathbf{0}} = c + a \cdot \mathbf{0} + b \cdot \mathbf{1} \overset{(7),(3),(6)}{=} c + b$. Note that the the first $=$ sign is to show what returns the $\overline{\ }$ function, where the second $=$ sign represents the equality over actions from the $\mathcal{CAT}$ algebra. Obviously $c + b + b\&c \neq c + b$ which ends our proof. □

The following result proves some properties of the action negation we have. The same properties hold for the action negation of [Mey88].

Proposition 2.15 (properties of action negation)**.** *The following hold:*

$$\overline{\overline{\alpha}} = \alpha \tag{25}$$

$$\overline{\alpha \cdot \beta} = \overline{\alpha} + \alpha \cdot \overline{\beta} \tag{26}$$

Proof: We prove first property (25). Consider $\alpha = +_{i \in I}\ \alpha^i_{\&} \cdot \alpha^i$ in canonic form. Then the negation $\overline{\alpha}$ is:

$$\begin{array}{rcl} \overline{\alpha} & = & +_{\gamma \in \overline{R}} \gamma\ +\ +_{i \in I}\ \alpha^i_{\&} \cdot \overline{\alpha^i} \\ & = & +_{\gamma \in \overline{R}I} \gamma \cdot \alpha_\gamma \text{ where } \overline{R}I = \overline{R} \cup \{\alpha^i_{\&} \mid i \in I\} \text{ and} \\ & & \qquad \alpha_\gamma = \mathbf{1} \text{ if } \gamma \in \overline{R} \\ & & \qquad \alpha_\gamma = \overline{\alpha^i} \text{ if } \gamma \in \{\alpha^i_{\&} \mid i \in I\} \end{array}$$

All we did above is just to rewrite the $\overline{\alpha}$ in a notation which would help us to reason easier further. Consider now $\overline{\overline{\alpha}}$ which is:

$$\begin{array}{rcl} \overline{\overline{\alpha}} & = & \overline{+_{\gamma \in \overline{R}I} \gamma \cdot \alpha_\gamma} \\ & = & +_{\gamma \in \overline{\overline{R}I}} \gamma\ +\ +_{\gamma \in \overline{R}I} \gamma \cdot \overline{\alpha_\gamma} \end{array}$$

with $\overline{\overline{R}I} = \{\gamma \mid \gamma \in \mathcal{A}^{\&}_B, \text{ and } \forall \gamma' \in \overline{R}I,\ \gamma' \not\leq_{\&} \gamma\}$. It is easy to see that $\overline{\overline{R}I} = \emptyset$. Consider $\exists \gamma \in \overline{\overline{R}I}$ then by the definition of $\overline{\overline{R}I}$ it means that $\forall i \in I,\ \alpha^i_{\&} \not\leq_{\&} \gamma$. On the other hand this implies, by the definition of $\overline{R}$ that $\gamma \in \overline{R}$. This is a contradiction with the supposition that $\gamma \in \overline{\overline{R}I}$ because it is violates the condition that $\forall \gamma' \in \overline{R},\ \gamma' \not\leq_{\&} \gamma$.

Because $\overline{\overline{R}I} = \emptyset$ the double negation becomes:

$$
\begin{array}{lcl}
\overline{\overline{\alpha}} & = & +_{\gamma\in\overline{R}I}\,\gamma\cdot\overline{\alpha_\gamma} \text{ where } \overline{R}I = \overline{R}\cup\{\alpha^i_\& \mid i\in I\} \text{ and} \\
 & & \qquad\qquad \alpha_\gamma = \mathbf{1} \text{ if } \gamma\in\overline{R} \\
 & & \qquad\qquad \alpha_\gamma = \overline{\alpha^i} \text{ if } \gamma\in\{\alpha^i_\& \mid i\in I\} \\
 & = & +_{\gamma\in\overline{R}}\,\gamma\cdot\overline{\mathbf{1}} \;+\; +_{i\in I}\;\alpha^i_\&\cdot\overline{\overline{\alpha^i}} \\
 & = & +_{\gamma\in\overline{R}}\,\gamma\cdot\mathbf{0} \;+\; +_{i\in I}\;\alpha^i_\&\cdot\overline{\overline{\alpha^i}} \\
 & \overset{(7),(3)}{=} & +_{i\in I}\;\alpha^i_\&\cdot\overline{\overline{\alpha^i}}.
\end{array}
$$

By structural induction we know that $\overline{\overline{\alpha^i}} = \alpha^i$ and thus we finish the proof of property (25).

We continue by proving property (26). Consider $\alpha = +_{i\in I}\;\alpha^i_\&\cdot\alpha^i$ in canonic form, then:

$$
\begin{array}{lcl}
\alpha\cdot\beta & = & (+_{i\in I}\;\alpha^i_\&\cdot\alpha^i)\cdot\beta \\
 & \overset{(9)}{=} & +_{i\in I}\;\alpha^i_\&\cdot\alpha^i\cdot\beta.
\end{array}
$$

Therefore the negation is:

$$
\begin{array}{lcl}
\overline{\alpha\cdot\beta} & = & \overline{+_{i\in I}\;\alpha^i_\&\cdot\alpha^i\cdot\beta} \\
 & = & +_{\gamma\in\overline{R}}\,\gamma \;+\; +_{i\in I}\;\alpha^i_\&\cdot\overline{\alpha^i\cdot\beta}.
\end{array}
$$

On the other hand we have that:

$$
\begin{array}{lcl}
\overline{\alpha}+\alpha\cdot\overline{\beta} & = & +_{\gamma\in\overline{R}}\,\gamma \;+\; +_{i\in I}\;\alpha^i_\&\cdot\overline{\alpha^i} \;+\; +_{i\in I}\;\alpha^i_\&\cdot\alpha^i\cdot\overline{\beta} \\
 & = & +_{\gamma\in\overline{R}}\,\gamma \;+\; +_{i\in I}\;\alpha^i_\&\cdot(\overline{\alpha^i} \;+\; \alpha^i\cdot\overline{\beta}).
\end{array}
$$

By induction on the structure of α we have that $\overline{\alpha^i\cdot\beta} = \overline{\alpha^i} \;+\; \alpha^i\cdot\overline{\beta}$ and thus the proof is finished.

Note: The negation of $\alpha\cdot\beta$ behaves as follows. It first unfolds by negation the action α until it reaches the ends of each branch; i.e. reaches $\alpha^i = \mathbf{1}$. At the end of each branch of α it is found β and thus the second member of the choice; i.e. $\alpha\cdot\overline{\beta}$. One thing remains to notice is that the negation of α contains some branches which are exacly the branches of α but ending in a $\mathbf{0}$, and these branches are not contained in the negation of $\alpha\cdot\beta$. Even so, by applying the axioms (7) and (3) these extra branches dissapear. □

After reading the proof above we observe easily the following corollary which explains how the action and its negation should be viewd *complementary* (notion which becomes more clear after reading the proofs of Propositions 3.11 and 3.12).

Corollary 2.16. *There always exists an action equal to the negation $\overline{\alpha}$ which does not contain any full path of the negated action α. That is because when-*

$$
\begin{array}{rcl}
\mathcal{C}ontract & := & \mathcal{D}\ ;\ \mathcal{C} \\
\mathcal{C} & := & \mathcal{C}_O \mid \mathcal{C}_P \mid \mathcal{C}_F \mid \mathcal{C} \wedge \mathcal{C} \mid [\beta]\mathcal{C}' \mid \langle\beta\rangle\mathcal{C}' \mid \top \mid \ \bot \\
\mathcal{C}' & := & \phi \mid \mathcal{C} \\
\mathcal{C}_O & := & O_{\mathcal{C}}(\alpha) \mid \mathcal{C}_O \oplus \mathcal{C}_O \\
\mathcal{C}_P & := & P(\alpha) \mid \mathcal{C}_P \oplus \mathcal{C}_P \\
\mathcal{C}_F & := & F_{\mathcal{C}}(\alpha) \mid \mathcal{C}_F \vee [\alpha]\mathcal{C}_F \\
\alpha & := & \mathbf{0} \mid \mathbf{1} \mid a \mid \alpha\&\alpha \mid \alpha \cdot \alpha \mid \alpha + \alpha \\
\beta & := & \alpha \mid \beta^* \mid \mathcal{C}?
\end{array}
$$

Table 4: $\mathcal{CL}$ syntax including the actions.

ever we reach the end of a path in the original action it means that we reach a $\mathbf{1}$ *action as* α^i*. When negating* $\overline{\alpha^i} = \overline{\mathbf{1}} = \mathbf{0}$ *that branch ends in* $\mathbf{0}$ *which may propagate upwords by axiom (7)* $\alpha \cdot \mathbf{0} = \mathbf{0}$ *and may dissapear eventually by axiom (3)* $\alpha + \mathbf{0} = \alpha$*.*

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 we gave for this purpose. In this section we give two direct semantics for the $\mathcal{CL}$ language using the $\mathcal{CAT}$ algebra.

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[14] 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 section ends by studying the relation between the two semantics.

We have seen the actions studied in the $\mathcal{CA}$ algebra. By now we have a more clear view over the syntax of the $\mathcal{CL}$ language as we know exacty how the actions α are constructed. More, we give in Table 4 also the definition of the actions β from PDL.

[14]The linear semantics is more restricted in terms of expressivety and does not capture what the branching semantics does.

3.1 Branching semantics in terms of normative structures

We start by defining some preliminary notions. First we give the definition of a *labeled Kripke structure* that we use.[15]

Definition 3.1 (Labelled Kripke Structure)**.** *A labeled Kripke structure is a structure* $K = (W, R_{2^{\mathcal{A}_B}}, \mathcal{V})$ *where* W *is a set of worlds (states),* $\mathcal{V} : P \to 2^W$ *is a valuation function of the propositional constants returning a set of worlds where the constants hold.* $\mathcal{A}_B$ *is a finite set of basic labels (called basic actions),* $2^{\mathcal{A}_B}$ *is the set of sets built with the elements of* $\mathcal{A}_B$*, and* $R_{2^{\mathcal{A}_B}} : 2^{\mathcal{A}_B} \to 2^{W \times W}$ *is a function returning for each concurrent action (i.e. set of basic actions) a set of pairs of worlds (intuitively* $R_{2^{\mathcal{A}_B}}$ *gives a relation on the worlds for each set label).*

The rooted trees and the guarded rooted trees are defined as in sections 2.2.1 and 2.4 respectively. We use the notation T_n to denote the subtree of T with root in the node n of T. Note that we are working only with *pruned* trees as in Definition 2.9 which have far less τ labels and black nodes (see Proposition 2.4). For brevity we always denote by an indexed t a node of a tree and by an indexed k a world of a labeled Kripke structure. Henceforth we use the more graphical notation $t \xrightarrow{\alpha} t'$ ($k \xrightarrow{\alpha} k'$) for an edge (transition) in a tree (Kripke structure) instead of the classical one (t, α, t') ($(k, k') \in R_{2^{\mathcal{A}_B}}(\alpha)$) that we had before. Note that we consider both the guarded rooted trees and the labelled Kripke structures to have the same set of basic labels (actions) $\mathcal{A}_B$.

Definition 3.2. *For a rooted tree* $T = (\mathcal{N}, E, \mathcal{A})$ *and a labeled Kripke structure* $K = (W, R_{2^{\mathcal{A}_B}}, \mathcal{V})$ *we define a relation* $\mathcal{S} \subseteq \mathcal{N} \times W$ *which we call the simulation of the tree node by the structure node.*

$$t \,\mathcal{S}\, k \;\; \textit{iff} \;\; \begin{array}{l} \forall t \xrightarrow{\gamma} t' \,|\, t, t' \in \mathcal{N},\, \exists k' \in W \;\; s.t. \;\; k \xrightarrow{\gamma'} k' \wedge (\gamma = \gamma' \vee \gamma <_{\&} \gamma') \\ \textit{and} \quad t' \,\mathcal{S}\, k' \end{array}$$

The simulation relation on nodes and worlds relates one node t of a tree with one world k of a Kripke structure on the basis that all the edges (i.e. actions) from the tree node t can be *simulated* by a transition (i.e. action) from the world k. Moreover, the resulting node and world after the action is executed are also in the same relation. Note that the label of the transition in K may be more demanding than the label of the edge in the tree T. We may strengthen this condition:

[15]The definition is standard, but several definitions of Kripke structures exist in the literature and we want to avoid confusion.

Definition 3.3. *A strong simulation is denoted by* $\mathcal{S}^s$ *and in the same conditions as before, it is defined as:*
$t\,\mathcal{S}^s\,k$ *iff* $\forall t \xrightarrow{\gamma} t' \,|\, t, t' \in \mathcal{N}$, $\exists k' \in W$ *s.t.* $k \xrightarrow{\gamma} k'$ *and* $t'\,\mathcal{S}^s\,k'$

Note that the simulation relation requires that all the edges starting at the tree node t are simulated by a transition in the Kripke structure. We can weaken this condition:

Definition 3.4. *A partial simulation is denoted by* $\tilde{\mathcal{S}}$ *and in the same conditions as before, it is defined as:*
$t\,\tilde{\mathcal{S}}\,k$ *iff* $\exists t \xrightarrow{\gamma} t' \,|\, t, t' \in \mathcal{N}$, *s.t.* $\exists k' \in W$ *s.t.* $k \xrightarrow{\gamma'} k' \wedge (\gamma = \gamma' \vee \gamma <_{\&} \gamma')$ *and* $t'\,\tilde{\mathcal{S}}\,k'$

The partial simulation may be extended to a strong variant in the same way as in Definition 3.3. We may now extend the simulations before to relate trees and structures.

Definition 3.5 (Simulation for rooted trees)**.** *We say that a rooted tree* $T = (\mathcal{N}, E, \mathcal{A})$*, with root* r *is simulated by a labeled Kripke structure* $K = (W, R_{2^{\mathcal{A}_B}}, \mathcal{V})$ *with respect to a state* i *of* K*, denoted* $T\ \mathcal{S}_i\,K$*, iff* $r\,\mathcal{S}\,i$*.*

Definition 3.6 (Partial simulation for trees)**.** *We say that a rooted tree* $T = (\mathcal{N}, E, \mathcal{A})$ *with root* r *is partially simulated by a labeled Kripke structure* $K = (W, R_{2^{\mathcal{A}_B}}, \mathcal{V})$ *with respect to a state* i *of* K*, denoted* $T\ \tilde{\mathcal{S}}_i\,K$*, iff* $r\,\tilde{\mathcal{S}}\,i$*.*

The extension of the definitions before to *guarded* rooted trees, i.e. where the nodes are labeled with guards ϕ is natural. We show below the extension only for the simulation relation over nodes; the other extensions are done analogue. For the conciseness of the notation we keep the same simbols for the extended versions of the relations. Moreover, we do not use explicitely the guard of the tree nodes when that guard is not important (but one may understand that a guard exists for that particular node).

Definition 3.7. *For a guarded rooted tree* $T = (\mathcal{N}, E, \mathcal{A})$ *and a labeled Kripke structure* $K = (W, R_{2^{\mathcal{A}_B}}, \mathcal{V})$ *we extend the relation* $\mathcal{S} \subseteq \mathcal{N} \times W$ *to labeled tree nodes.*
$t : \{\phi\}\,\mathcal{S}\,k$ *iff* $k \in \mathcal{V}(\phi)$ *and* $\forall t \xrightarrow{\gamma} t' \,|\, t, t' \in \mathcal{N}$, $\exists k' \in W$ *s.t.* $k \xrightarrow{\gamma'} k' \wedge (\gamma = \gamma' \vee \gamma <_{\&} \gamma')$ *and* $t' : \{\phi'\}\,\mathcal{S}\,k'$

Definition 3.8 (Simulation for guarded rooted trees)**.** *We say that a guarded rooted tree* $T = (\mathcal{N}, E, \mathcal{A})$ *with root* r *is simulated by a labeled Kripke structure* $K = (W, R_{2^{\mathcal{A}_B}}, \mathcal{V})$ *with respect to a state* i *of* K*, denoted* $T\mathcal{S}_iK$*, iff* $r : \{\phi\}\,\mathcal{S}\,i$*.*

Simulation of a labeled Kripke structure by a guarded rooted tree (i.e. the other direction of the simulation) is not possible to give, as in structures there may be cycles, wheres the finite trees have not. We see later that the simulation we have defined is enough for giving the semantics of the $\mathcal{CL}$ operators.

Definition 3.9 (maximal simulating structures)**.** *Whenever $T \mathcal{S}_i K$, a tree T is simualted by a structure K w.r.t. a state i then we call the* maximal simulating structure w.r.t. T and i*, and denote it by $K^{T,i}_{max} = (W', R'_{2^{\mathcal{A}_B}}, \mathcal{V}')$ the sub-structure of $K = (W, R_{2^{\mathcal{A}_B}}, \mathcal{V})$ s.t.:*

1. $i \in W'$
2. $\mathcal{V}' = \mathcal{V}|_{W'}$
3. $\forall t \xrightarrow{\gamma} t' \in T$ *then* $\forall k \xrightarrow{\gamma'} k' \in K$ *s.t.* $t \mathcal{S} k \wedge (\gamma <_{\&} \gamma' \vee \gamma = \gamma')$ *do add* k' *to* W' *and add* $k \xrightarrow{\gamma'} k'$ *to* $R'_{2^{\mathcal{A}_B}}$.
4. *nothing else is in* $K^{T,i}_{max}$

We call the non-simulating reminder of K w.r.t. T and i *the sub-structure* $K^{T,i}_{rem} = K - K^{T,i}_{max}$.

We consider a slight variation of a Kripke structure which we call *normative structure* and usually denote by $K^{\mathcal{N}}$. All the definitions of simulation that we defined above hold without change on normative structures. The notations are also extended naturally to the new notation.

Definition 3.10 (Normative structure)**.** *A normative structure is a labeled Kripke structure as in Definition 3.1 with the following extensions:*

- *There is a set P_c of special propositional constants O_a and $\mathcal{F}_a$ indexed by the atomic actions of $\mathcal{A}_B$ ($a \in \mathcal{A}_B$);*
- *The transitions are deterministic; i.e. the function $R_{2^{\mathcal{A}_B}}$ associates to each label a partial function instead of a relation, therefor for each label from one world there is at most one reachable world.*

The deterministic restriction of our normative structure has been investigated in [BAHP81] as the restriction of PDL to a deterministic structure and there an EXPTIME decision procedure is given for the satisfiability problem. The axiom that encodes the determinism of the structure is:

$$\langle a \rangle \mathcal{C} \Rightarrow [a]\mathcal{C}, \quad \forall a \in \mathcal{A}_B$$

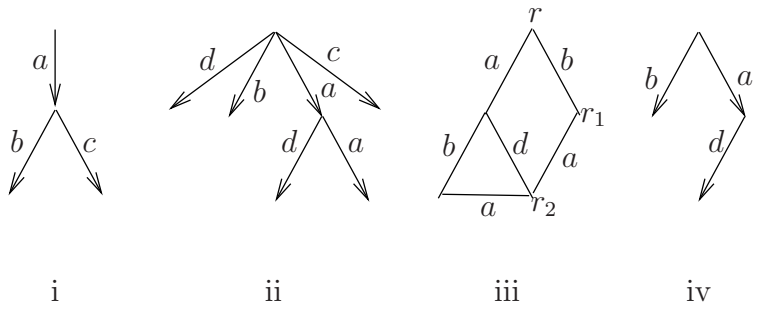

Figure 7: Examples of finite rooted trees with labeled edges.

We present now a direct semantics for $\mathcal{CL}$ in terms of satisfiability w.r.t. a model and a state. Our model is the normative structure $K^{\mathcal{N}}$.

The semantics of the assertions ϕ, or the dynamic modalities $[\cdot]$ and $\langle\cdot\rangle$ are the classical ones.

$K^{\mathcal{N}}, i \models \phi$ iff $i \in \mathcal{V}(\phi)$

$K^{\mathcal{N}}, i \models [\beta]\mathcal{C}$ iff $\forall k \in W$ s.t. $(i,k) \in R_{2^{\mathcal{A}_B}}(\beta)$ then $K^{\mathcal{N}}, k \models \mathcal{C}$.

For $\langle\beta\rangle\mathcal{C}$ we just need to change the $\forall k$ into $\exists k$. Recall from dynamic logic that compound actions β define relations over the worlds of the Kripke structure starting from the partial functions given for the basic actions by the $R_{2^{\mathcal{A}_B}}$ using relation union $\cup$ for choice $+$, relation composition $\circ$ for sequence $\cdot$ and reflexive and transitive closure * for the Kleene star.

Example: Consider the following example for action negation inside the dynamic modality: take the set of basic actions $\mathcal{A}_B = \{a, b, c, d\}$, the compound action $\alpha = a \cdot (b + c)$ with the tree of Figure 7-i, and the labeled Kripke structure $K^{\mathcal{N}}$ of Figure 7-iii. The action negation $\overline{\alpha}$ has the tree in Figure 7-ii. We want to check if $[\overline{\alpha}]\mathcal{C}$ holds in the structure $K^{\mathcal{N}}$ at state r, i.e. $K^{\mathcal{N}}, r \models [\overline{\alpha}]\mathcal{C}$. The relation defined by $\overline{\alpha}$ is $\{d, b, ad, aa, c\}$ (instead of a pair of nodes we write the corresponding sequence of labels through the tree and consider the pair of the nodes at the begining and the end of the sequence). All the nodes from the Kripke structure that are in this relation are $\{b, ad\}$ (i.e. r_1 and r_2). In order for the formula to hold in the structure at node r it remains that in all these nodes the clause $\mathcal{C}$ should hold; essentialy this means that $K^{\mathcal{N}}, r_1 \models \mathcal{C}$ and $K^{\mathcal{N}}, r_2 \models \mathcal{C}$.

The propositional connectives have the classical semantics:

$K^{\mathcal{N}}, i \models \mathcal{C}_1 \wedge \mathcal{C}_2$ iff $K^{\mathcal{N}}, i \models \mathcal{C}_1$ and $K^{\mathcal{N}}, i \models \mathcal{C}_2$

For the connectives $\vee$ respectively $\oplus$ the semantics is similar to the one for $\wedge$ where we change the word *and* to respectively *or*, *xor*.

More interesting and particular to our $\mathcal{CL}$ logic is the interpretation of the deontic modalities.

$K^{\mathcal{N}}, i \models O_{\mathcal{C}}(\alpha)$ iff $\hat{I}_{\mathcal{CA}}(\alpha)\mathcal{S}_i K^{\mathcal{N}}$ and

$\forall t \xrightarrow{\gamma} t' \in \hat{I}_{\mathcal{CA}}(\alpha)$, $\forall k \xrightarrow{\gamma'} k' \in K^{\mathcal{N}}$ s.t. $t\,\mathcal{S}\,k \wedge (\gamma <_{\&} \gamma' \vee \gamma = \gamma')$
then $\forall a \in \mathcal{A}_B$ if $a \in \gamma$ then $k' \in \mathcal{V}(O_a)$ and

$\forall k \xrightarrow{\gamma'} k' \in K^{\hat{I}_{\mathcal{CA}}(\alpha),i}_{rem}$
then $\forall a \in \mathcal{A}_B$ if $a \in \gamma'$ then $k' \notin \mathcal{V}(O_a)$

and $K^{\mathcal{N}}, i \models [\overline{\alpha}]\mathcal{C}$.

Note that because $K^{\mathcal{N}}$ simulates the pruned tree $\hat{I}_{\mathcal{CA}}(\alpha)$ corresponding to the action α under the obligation, we are guaranteed that for each edge $t \xrightarrow{\gamma} t'$ in the tree there exists at leas one corresponding transition in the structure.

Notations: we have in the $\mathcal{CL}$ language the two special propositions $\top$ and $\bot$ (i.e. *true* and *false*); we denote by just $O(\alpha)$ the $O_{\top}(\alpha)$ which basically means that there is no reparation associated with the obligation. The semantics of $O(\alpha)$ is the semantics of $O_{\top}(\alpha)$ and because $[\overline{\alpha}]\top$ is true in all the states then the last line in the semantical definition of obligation can be ignored.

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 executed) then the reparation $\mathcal{C}$ should hold. The first part of the semantics is the interpretation of the obligation. The structure $K^{\mathcal{N}}$ must simulate the tree of the action α; note that it is not strong simulation, because in the structure there may be transitions labeled with more demanding actions (eg. $a\&b\&c$ is more demanding than a) which intuitively if we execute these actions then the obligation in α is still respected. The second and third lines formalize the fact that for all the transitions of the structure which simulate edges in the tree it must be that in the ending states k' there must be one proposition O_a for each basic action that makes the label of the simulated edge of the tree. Lines four and five of the semantics interpret the fact that on any other edges of the structure appear no proposition O_a.

We see how each of the particularities of this semantics helps in provind the properties of Lemma 3.6 and in proving the Proposition 3.1.

Fair obligations: We add to the above semantics for obligation a *fairness constraint* to obtain what we call *fair obligations*. Consider the tree unfolding of the minimal simulating structure w.r.t. a tree $\hat{I}_{\mathcal{CA}}(\alpha)$ of an action α, and denote that by $TK_{max}^{\hat{I}_{\mathcal{CA}}(\alpha),}$. Note that the unfolding may give an infinite tree. We add the fairness condition (27) below to the definition of the semantics of obligation so to get fair obligations:

$$\exists\beta \text{ s.t. } \hat{I}_{\mathcal{CA}}(\alpha) \parallel \hat{I}_{\mathcal{CA}}(\beta) \doteq TK_{max}^{\hat{I}_{\mathcal{CA}}(\alpha),} \tag{27}$$

Because of the completeness of the algebra of actions w.r.t. the interpretation as guarded rooted trees we can give an alternative characterization of the fairness condition above. The characterization is in terms of actions. Without loss of generality we can restrict the infinite tree $TK_{max}^{\hat{I}_{\mathcal{CA}}(\alpha),}$ to a finite one by removing all the nodes which are below a depth equal to the height of the tree interpreting α (i.e. $\hat{I}_{\mathcal{CA}}(\alpha)$). Take the action α which is characteristic for the tree $\hat{I}_{\mathcal{CA}}(\alpha)$ and the action γ which is the action corresponding to the restricted finite tree $TK_{max}^{\hat{I}_{\mathcal{CA}}(\alpha),}$. Condition (27) is written as: $\exists\beta$ s.t. $\alpha\&\beta = \gamma$. We are working on the term algebra at the syntactic level and thus we can consider the condition above in a term rewriting system given by the axioms of the $\mathcal{CAT}$ algebra. In term rewiting the condition involves a variable X in place of the β. Thus we have the equality $\alpha\&X = \gamma$ which is known as the syntactic unification problem which is undecidable in general.

We are interested in having decidability for this problem (i.e. for our initial fairness condition).

Absorbtion constraint: We add to the fair obligations another constraint which guarantes that the maximal simulating Kripke structure does not absorbe the tree of the action α (the action that is obligatory). We say that a fair obligation respects the absorbtion constraint iff it the following is the case:

$$\alpha \not<^{ab}_{\&} \beta$$

If the absorbtion constraint is not satisfied then it might be the case that the β given by the fairness constraint absorbes the α; i.e. $\alpha\&\beta = \beta$ like in the case of $(a + b)\&\beta = a\&b$ where β returned by the fairness constraint is $(a\&b)$. We see later where the absorbtion constraint is necessary.

Notations: we use (as in dynamic logic [HTK00]) $p \Rightarrow q$ for $[p?]q$; in the $\mathcal{CL}$ language we have $\mathcal{C}_1 \Rightarrow \mathcal{C}_2$ for $[\mathcal{C}_1?]\mathcal{C}_2$. We use $\mathcal{C}_1 \Leftrightarrow \mathcal{C}_2$ for $\mathcal{C}_1 \Rightarrow \mathcal{C}_2 \wedge \mathcal{C}_2 \Rightarrow \mathcal{C}_1$. As is often the case in tableau reasoning we denote negation of formulas $\neg O(a)$ using implication $O(a) \Rightarrow \bot$.

$K^{\mathcal{N}}, i \models F(\alpha)$ iff whenever $\hat{I}_{\mathcal{CA}}(\alpha)\mathcal{S}_i^{\sim} K^{\mathcal{N}}$ then,
$\forall T$ a subtree of $\hat{I}_{\mathcal{CA}}(\alpha)$ s.t. $T\mathcal{S}_i K^{\mathcal{N}}$, and $\forall \sigma$ a branch in T
$\exists t \xrightarrow{\gamma} t' \in \sigma$ s.t.
$\forall k \xrightarrow{\gamma'} k'$ with $t\,\mathcal{S}\,k \wedge t'\,\mathcal{S}\,k'$
then $\forall a \in \mathcal{A}_B$ if $a \in \gamma'$ then $k' \in \mathcal{V}(\mathcal{F}_a)$.

For the F modality we use partial simulation $\mathcal{S}_r^{\sim}$ between the tree and the normative structure in order to have our intuition that if an action is not present as a label of an outgoing transition of the model then the action is *by default* considered forbidden. In the second line we consider *all* subtrees and for each of them *all* branches 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 a node on each path in order to respect the intuition that $F(a \cdot b) = F(a) \vee \langle a \rangle F(b)$, forbidding a sequence means forbidding some action on that sequence. In the last two lines of the semantics of F we look for all the transitions of the normative structure from the chosen node which have a label *more demanding* than the label of the tree; this is in order to respect the intuition that $F(a) \Rightarrow F(a\&b)$, forbidding an action implies forbidding any action more demanding. The semantics of O, P, or F relates to the trace-based semantics of Process Logic [Pra79] and to some extent to the modalities of [VdM90].

$K^{\mathcal{N}}, i \models P(\alpha)$ iff $\hat{I}_{\mathcal{CA}}(\alpha)\mathcal{S}_i^s K^{\mathcal{N}}$, and
$\forall t \xrightarrow{\gamma} t' \in \hat{I}_{\mathcal{CA}}(\alpha)$, $\forall k \xrightarrow{\gamma} k' \in K^{\mathcal{N}}$ s.t. $t\,\mathcal{S}^s\,k \wedge t'\,\mathcal{S}^s\,k'$
then $\forall a \in \mathcal{A}_B$ if $a \in \gamma$ then $k' \in \mathcal{V}(\neg\mathcal{F}_a)$.

Note that for O, P, or F the semantic interpretation "walks" through the nodes of the whole tree of the action, where in the case of $[\beta]$ it has to look only at the nodes at the boundary of the tree (the leaf nodes).

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.

We say that a contract $\mathcal{C}$ is *satisfied in a normative structure* $K^{\mathcal{N}}$ *at a state* w iff $K^{\mathcal{N}}, w \models \mathcal{C}$. We say that a contract $\mathcal{C}$ is *satisfiable w.r.t. a normative structure* $K^{\mathcal{N}}$, and denote it by $K^{\mathcal{N}} \models \mathcal{C}$ iff $\forall w \in K^{\mathcal{N}}$ then $K^{\mathcal{N}}, w \models \mathcal{C}$. 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 for contracts is interesting as informally it models that fact that given a contract we ask if there exists some parties (which are the ones in the contract) with some specific behavior that would 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.

We say that a contract is *valid* and denote it by $\models \mathcal{C}$ iff the contract is satisfiable in any model; i.e. $\forall K^{\mathcal{N}},\ K^{\mathcal{N}} \models \mathcal{C}$. At an intuitive understanding, 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.

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 contracts 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 $\mathcal{C}$.

The validity problem for contracts is not so interesting. 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*.

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 automata*. If we take as models just the subset of normative automata which do not contain violating runs then we talk about *strict satisfiability*. This means that if a contract is strictly satisfiable then the contract cannot be violated; i.e. the contract must be respected in the given model.

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* 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

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 3.1 (Properties of obligations)**.**

$$O_{\mathcal{C}}(a) \wedge O_{\mathcal{C}}(b) \Rightarrow O_{\mathcal{C}}(a\&b) \tag{28}$$
$$O_{\mathcal{C}}(a) \not\models O_{\mathcal{C}}(a\&b) \tag{29}$$
$$O_{\mathcal{C}}(a+b) \not\models O_{\mathcal{C}}(a\&b) \tag{30}$$
$$O_{\mathcal{C}}(a\&b) \not\models O_{\mathcal{C}}(a+b) \tag{31}$$
$$O_{\mathcal{C}}(a+b) \not\models O_{\mathcal{C}}(a) \tag{32}$$
$$O_{\mathcal{C}}(a) \not\models O_{\mathcal{C}}(a+b) \tag{33}$$
$$O_{\mathcal{C}}(a+b) \not\models O_{\mathcal{C}}(a) \oplus O_{\mathcal{C}}(b) \tag{34}$$
$$O_{\mathcal{C}}(a) \oplus O_{\mathcal{C}}(b) \not\models O_{\mathcal{C}}(a+b) \tag{35}$$
$$O_{\mathcal{C}}(a\&b) \wedge O_{\mathcal{C}}(a+b) \Rightarrow O_{\mathcal{C}}(a\&b) \tag{36}$$

Proof: Proof of (30) follows from (32). Proof of (36) is known from

classical propositional logic, and it can also be deduced from (28). Proof of (29) follows from (28).

- *Proof of property (28):*

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. We suppose that there exist two obligations enforced at the same time, say $O_{\mathcal{C}}(\alpha)$ and $O_{\mathcal{C}}(\beta)$, where α and β are different compound actions; i.e. $\exists K$ a normative structure and $\exists w \in W$ a world in K s.t. $K, w \models O_{\mathcal{C}}(\alpha) \wedge O_{\mathcal{C}}(\beta)$. We have to prove that the structure satisfies the obligation on the right; i.e. $K, w \models O_{\mathcal{C}}(\alpha \& \beta)$.

We give first a series of helper results.

Lemma 3.2. *If* $K, w \models O_{\mathcal{C}}(\alpha) \wedge O_{\mathcal{C}}(\beta)$ *then* $K_{max}^{\hat{I}_{\mathcal{CA}}(\alpha),w} = K_{max}^{\hat{I}_{\mathcal{CA}}(\beta),w}$.

Proof: If $K, w \models O_{\mathcal{C}}(\alpha) \wedge O_{\mathcal{C}}(\beta)$ then $K, w \models O_{\mathcal{C}}(\alpha)$ and $K, w \models O_{\mathcal{C}}(\beta)$. From the first we have by the semantics that $\hat{I}_{\mathcal{CA}}(\alpha)\ \mathcal{S}_w\ K$ which means that there exists the maximal simulating structure $K_{max}^{\hat{I}_{\mathcal{CA}}(\alpha),w}$. From the semantics of $O_{\mathcal{C}}(\beta)$ we obtain similarly $K_{max}^{\hat{I}_{\mathcal{CA}}(\beta),w}$. Both maximal simulating structures are substructures of the same K. We use the proof principle *reductio ad absurdum* and suppose that $K_{max}^{\hat{I}_{\mathcal{CA}}(\alpha),w} \neq K_{max}^{\hat{I}_{\mathcal{CA}}(\beta),w}$ then (without loss of generality we suppose that) $K_{max}^{\hat{I}_{\mathcal{CA}}(\alpha),w} \setminus K_{max}^{\hat{I}_{\mathcal{CA}}(\beta),w} = K' \neq \emptyset$. This means that $K' \cap K_{max}^{\hat{I}_{\mathcal{CA}}(\beta),w} = \emptyset$ and $K' \subseteq K_{rem}^{\hat{I}_{\mathcal{CA}}(\beta),w}$ and by the semantics of $O_{\mathcal{C}}(\beta)$ we know that $\forall k \xrightarrow{\gamma'} k' \in K'$ then $\forall a \in \mathcal{A}_B$ if $a \in \gamma'$ then $k' \notin \mathcal{V}(O_a)$. On the other hand $K' \subseteq K_{max}^{\hat{I}_{\mathcal{CA}}(\alpha),w}$ which by the semantics of $O_{\mathcal{C}}(\alpha)$ means that $\forall k \xrightarrow{\gamma'} k' \in K'$ then $\forall a \in \mathcal{A}_B$ if $a \in \gamma$ then $k' \in \mathcal{V}(O_a)$. This results in a contradiction and thus we have a contradiction with our initial supposition forcing us to conclude that $K_{max}^{\hat{I}_{\mathcal{CA}}(\alpha),w} = K_{max}^{\hat{I}_{\mathcal{CA}}(\beta),w}$. □

Lemma 3.3. *For any pruned trees that satisfy the following:* $T_1 \parallel T_2 = T$ *and* $T_3 \parallel T_4 = T$ *then it is the case that* $T = T_1 \parallel T_3 \parallel T_5$.

We would like to use a more familiar notation instead of the general one from the statement of the lemma so we rewrite it as: for any trees $T_\alpha \parallel T_{\gamma'} = T$ and $T_\beta \parallel T_{\gamma''} = T$ then we have $T = T_\alpha \parallel T_\beta \parallel T_{\gamma'''}$. This general presentation can be then particularized as: $\hat{I}_{\mathcal{CA}}(\alpha) \parallel T' = T$ and $\hat{I}_{\mathcal{CA}}(\beta) \parallel T'' = T$ then we have $T = \hat{I}_{\mathcal{CA}}(\alpha) \parallel \hat{I}_{\mathcal{CA}}(\beta) \parallel T'''$. This particular instance of the lemma involves pruned trees. There might appear special

cases from the application of the pruning procedure when the trees of the actions α and β contain some black nodes or some τ labels on the edges (which come from the fact that the actions contain in their composition occurrences of the special actions $\mathbf{1}$ or $\mathbf{0}$). The proof of the special cases require special care for the details in the application of the $\|$ operator and the pruning procedure. We leave these for the interested reader.

Proof: We will use the more familiar notation and sometimes we oversimplify the notation for the sake of a more clear and succint presentation. The proof is based on the definition of the $\|$ operator from Section 2.2.1. Note that the definition of $\|$ works on one level of the trees at a time; that is, it first deals with the edges on the first levels of the two trees and builds from these the first level of the resulting tree; afterwards it goes on to the lower levels. Therefore, w.l.o.g. we can look only at the first levels of the trees. Take $\alpha_1, \alpha_2, \ldots, \alpha_k$ to be the labels of the edges on the first level of the tree T_α; note that there are k edges. Take $\beta_1, \beta_2, \ldots, \beta_l$ the labels of the first level of the tree T_β.

To prove the lemma we use the proof principle *reductio ad absurdum* and suppose that $T \neq T_\alpha \parallel T_\beta \parallel T'''$ where the tree T''' is an arbitrary tree. The supposition is equivalent to saying that the labels on the first level of the tree T (we will denote these labels of T by τ_i) are not constructed from the labels of the tree $T_\alpha \parallel T_\beta$. This may come from several reasons.

First, it may be that none of the labels τ_i on the first level of the tree T contain one of the labels on the first level of $T_\alpha \parallel T_\beta$;[16] i.e. say the label $\alpha_1 \cup \beta_1$ is not contained in any of the τ_i.

Take n' and n'' to be the number of labels on the first level of the tree T' respectively T''. From the hypothesis $T_\alpha \parallel T_{\gamma'} = T$ of the lemma we know that there exit n' labels on the first level of the tree T which contain α_1; we call them $\tau^i_{\alpha_1}$ with $0 < i \leq n'$. Similarly from the hypothesis $T_\beta \parallel T_{\gamma''} = T$ we know that β_1 appears n'' times in the labels of T which we denote by $\tau^j_{\beta_1}$. On the other hand, from the supposition we know that β_1 does not appear in any of the $\tau^i_{\alpha_1}$ labels; and similarly α_1 does not appear in any of the $\tau^j_{\beta_1}$. Also from the hypothesis $T_\beta \parallel T_{\gamma''} = T$ we know that in all τ labels of T it appears one of the β_j labels. This meant that in each of the labels $\tau^i_{\alpha_1}$ it appears one of the β_j where $j \neq 1$. W.l.o.g. we consider one of these labels $\tau^1_{\alpha_1} = \alpha_1\beta_2\gamma$ for some γ which may very well be empty. From the same hypothesis $T_\beta \parallel T_{\gamma''} = T$ and knowing that $\alpha_1\beta_2\gamma$ is a label from T then it

[16] Recall that the labels on the first level of the tree $T_\beta \parallel T_\beta$ are $\alpha_1\beta_1, \ldots, \alpha_1\beta_l, \ldots, \alpha_k\beta_l$. Note that we sometimes just write $\alpha_1\beta_1$ instead of $\alpha_1 \cup \beta_1$. Some of these labels may be the same and the operator $\|$ combines them into on and makes the join of the two subtrees. This does not interfeer with the proof of the lemma.

means that $\alpha_1\gamma$ is a label on the first level of the tree T''. This implies that between the labels τ of T there exist each of the following labels $\alpha_1\gamma\beta_j$ with $j \neq 2$; i.e. the label $\alpha_1\gamma$ must be attached to each of the β_j labels of T_β including β_1.

Thus we obtain that there exists a label τ which contains $\alpha_1\beta_1$ which is a contradiction with the initial supposition. Therefore, each of the labels $\alpha_i\beta_j$ of the tree $T_\alpha \parallel T_\beta$ are contained in the labels τ of T.

A second way to contradict the lemma is to suppose that it is not the case that for all pairs $\alpha_i\beta_j$ there exits a same label γ such that $\alpha_i\beta_j\gamma = \tau$ is a label on the first level of the tree T. To explain it differently, this supposition wants to contradict the second $\parallel$ operator in the conclusion of the lemma $(T_\alpha \parallel T_\beta) \parallel T_{\gamma'''}$ which by the definition it must be that for each γ a label on the first level of the tree T''' it must be combined with each label $\alpha_i\beta_j$ of the tree $T_\alpha \parallel T_\beta$.

We take an arbitrary pair $\alpha_i\beta_j$, say $\alpha_1\beta_1$ and w.l.o.g. suppose it has some extra label γ which may be also empty. Thus $\alpha_1\beta_1\gamma$ is a label on the first level of the tree T. From the hypothesis $T_\beta \parallel T_{\gamma''} = T$ and knowing that β_1 is combined with the label $\alpha_1\gamma$ it implies that all other β_j with $1 < j \leq n''$ must be combined with the same label. Therefore, the following are also labels τ: $\alpha_1\beta_2\gamma, \ldots, \alpha_1\beta_l\gamma$. On the other hand, from the hypothesis $T_\alpha \parallel T_{\gamma'} = T$ and knowing that $\alpha_1\beta_1\gamma$ is a τ label it means that all other α_i labels must be combined with $\beta_1\gamma$. Therefore, we also have as τ labels: $\alpha_2\beta_1\gamma, \ldots, \alpha_k\beta_1\gamma$.

We continue to apply recursively the same reasoning on the new deduced labels like $\alpha_2\beta_1\gamma$ and we obtain in the end that all the labels $\alpha_k\beta_l$ appear among the labels τ on the first level of the tree T combined with the same label γ. Thus, the second false supposition is contradicted.

The last way of contradicting the lemma is trivial and it supposes that it is not the case that all the τ labels of T come from combination by $\parallel$ with the labels $\alpha_i\beta_j$. More clearly this tries to say that there exit other τ labels that do not follow the pattern decuced by the first two reasoning we had before. This cannot be as if there is another label besides $\alpha_i\beta_j\gamma$, say τ' we have proven by contradicting the first supposition that this must be of the form $\alpha_i\beta_j\gamma'$ and by the second supposition we again get that there exist all the $\alpha_i\beta_j\gamma'$ with $0 < i \leq n'$ and $0 < j \leq n''$ as labels τ on the first level of the tree T.

Thus the proof of the lemma is finished. □

Lemma 3.4. *For any actions α, β it is the case that*

$$\hat{I}_{\mathcal{CA}}(\alpha) \parallel \hat{I}_{\mathcal{CA}}(\beta) = \hat{I}_{\mathcal{CA}}(\alpha \& \beta).$$

Recall that $\hat{I}_{\mathcal{CA}}$ is not homomorphic with respect to the + operator (as argued on page 25) but the present lemma proves $\hat{I}_{\mathcal{CA}}$ to be homomorphic with respect to the $\|$ operator. Note also that we are working with pruned trees and we remind the reader of the characterization of pruned trees from Proposition 2.4.

Lemma 3.5. *For any K a normative structure and α, β two distinct actions we have that if $K, w \models O_{\mathcal{C}}(\alpha) \wedge O_{\mathcal{C}}(\beta)$ then $\hat{I}_{\mathcal{CA}}(\alpha \& \beta)\ \mathcal{S}_w\ K$.*

Proof: From the statement of the lemma $K, w \models O_{\mathcal{C}}(\alpha) \wedge O_{\mathcal{C}}(\beta)$ by applying the Lemma 3.2 we get that $K_{max}^{I_{\mathcal{CA}}(\alpha),w} = K_{max}^{I_{\mathcal{CA}}(\beta),w}$. This implies that the corresponding trees which unfold these maximal substructures are the same; i.e. $TK_{max}^{I_{\mathcal{CA}}(\alpha),w} = TK_{max}^{I_{\mathcal{CA}}(\beta),w} = TK_{max}$.

Moreover, from the hypothesis of the lemma we get that $K, w \models O_{\mathcal{C}}(\alpha)$ and $K, w \models O_{\mathcal{C}}(\beta)$. Considering the *fair obligations* constraint it implies that:

$$\exists T' \text{ s.t. } \hat{I}_{\mathcal{CA}}(\alpha) \parallel T' = TK_{max}^{I_{\mathcal{CA}}(\alpha),w}$$
$$\exists T'' \text{ s.t. } \hat{I}_{\mathcal{CA}}(\beta) \parallel T'' = TK_{max}^{I_{\mathcal{CA}}(\beta),w}$$

From these and knowing that the maximal simulating structures are the same we get that $\hat{I}_{\mathcal{CA}}(\alpha) \parallel T' = \hat{I}_{\mathcal{CA}}(\beta) \parallel T'' = TK_{max}$. By applying the Lemma 3.3 we get that $TK_{max} = \hat{I}_{\mathcal{CA}}(\alpha) \parallel \hat{I}_{\mathcal{CA}}(\beta) \parallel T'''$. By using Lemma 3.4 we know that $\hat{I}_{\mathcal{CA}}(\alpha) \parallel \hat{I}_{\mathcal{CA}}(\beta) = \hat{I}_{\mathcal{CA}}(\alpha \& \beta)$. Thus, $TK_{max} = \hat{I}_{\mathcal{CA}}(\alpha \& \beta) \parallel T'''$.

Following the Definition 3.5 of the simulation relation $\mathcal{S}_w$, in order to prove the conclusion $\hat{I}_{\mathcal{CA}}(\alpha \& \beta)\ \mathcal{S}_w\ K$ we need to prove that (1) $r\, \mathcal{S}\, w$ and (2) $\forall r \xrightarrow{\gamma} t' \in \hat{I}_{\mathcal{CA}}(\alpha \& \beta)$, $\exists w \xrightarrow{\gamma'} k' \in K$ s.t. $\gamma = \gamma' \vee \gamma <_{\&} \gamma'$ and $t'\, \mathcal{S}\, k'$. The proof of (1) is immediate from one of the hypothesis of the lemma (say from $\hat{I}_{\mathcal{CA}}(\alpha)\ \mathcal{S}_w\ K$).

Using the results above the proof of (2) becomes simple. As $\hat{I}_{\mathcal{CA}}(\alpha \& \beta) \parallel T''' = TK_{max}$ which is the tree unfolding of the substructure of K then it is simple to see that for any transition $r \xrightarrow{\gamma} t'$ of the tree $\hat{I}_{\mathcal{CA}}(\alpha \& \beta)$ there is a transition $w \xrightarrow{\gamma'} k' \in TK_{max}$ which clearly $\gamma = \gamma' \vee \gamma <_{\&} \gamma'$ depending on T'''. The fact that $t'\, \mathcal{S}\, k'$ is true is obvious by applying a similar recursive reasoning. □

□

Corollary 3.6. *At each moment in the execution of a contract there is one and only one obligation enforced over a complex action of the form $+_j(\&_i a_{ij})$.*

The statement of the lemma is the same to the following statement: At each state of a normative structure the combination of all the immediate

transitions define a unique obligation over the complex action defined by the corresponding subtree.

With the above semantics we have the following:

Proposition 3.7.

$$\textit{if } F(a) \textit{ then } F(a\&b) \tag{37}$$
$$F(a+b) \textit{ iff } F(a) \wedge F(b) \tag{38}$$
$$P(a+b) \textit{ iff } P(a) \wedge P(b) \tag{39}$$
$$F(a \cdot b) \textit{ iff } F(a) \vee \langle a \rangle F(b) \tag{40}$$
$$P(a \cdot b) \textit{ iff } P(a) \wedge [a] P(b) \tag{41}$$
$$\textit{is not the case that if } F(a\&b) \textit{ then } F(a) \tag{42}$$
$$\textit{is not the case that if } P(a\&b) \textit{ then } P(a) \tag{43}$$

Proof: All are easily proven by following the semantics above and the classical semantics for the propositional operators. For equation 50 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 3.8 (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 3.9 (finite model theorem)**.** *If a formula $\mathcal{C}$ has a model then it has a finite model.*

Related to Definition 2.14 we have the following.

Proposition 3.10. *The function* $[\ ]\mathcal{C} : \mathcal{CAT} \rightarrow \mathcal{CL}$ *is* independent of the representation *of the actions of* $\mathcal{CAT}$.

3.2 Linear semantics in terms of respecting traces

The present section is devoted to constructing a semantics for $\mathcal{CL}$ with the scope of monitoring enforcement of contracts. For this we are interested which actions are respecting the contract and which actions are violating the contract. More general which sequences of actions (called *traces* henceforth; in formalisms modelling programs a sequence of program actions is called an execution trace) are respecting the contract. We call a trace that respects a contract a *respecting* trace.

The roadmap is as follows: we first give a linear semantics for the expressions of $\mathcal{CL}$ in terms of respecting traces. This expresses the fact that a trace

respects (does not violate) a contract clause (expression of $\mathcal{CL}$). Obviously we do not capture everything that we do with the semantics based on normative structures of Section 3.1; Theorem 3.14 relates the two semantics. After having defined which traces do not violate a contract we construct a Büchi automaton for a contract clause which recognizes all the traces which do not violate the contract. This automaton is used to monitor the enforcement of a contract.

We continue by pin-pointing more formally what we understand by a trace. We follow the many works in the literature which have a presentation based on traces [Pra79, VdM90, Eme90, Maz88]. Consider a *trajectory* denoted $\sigma = (s_0, a_0, s_1, \ldots, a_m, s_{m+1})$ as an ordered sequence of subsequent alternation of *states* and *actions*. $m_\sigma \in \mathbb{N} \cup \infty$ is the length of a trajectory; note that trajectories may be infinite. When σ is obvious from the context we use just m instead of m_σ. We need to talk both about the states and about the actions of a trajectory. For this we use two projection functions: τ which returns just the sequence of actions $(a_0, \ldots, a_m)$ of a trajectory and π which returns the sequence of states $(s_0, \ldots, s_{m+1})$. We call τ *traces* and π *strings*. For both traces and strings we use ε to denote the empty trace or string (or sometimes the empty trajectory). We denote by $\sigma(i)$ an *instance* of a trajectory, by $\sigma(i..j)$ a finite *subtrajectory*, and by $\sigma(i..)$ the infinite subtrajectory starting at point i in σ (eg. $\pi(0)$ is the first state of a trajectory, $\tau(m)$ is the m^{th} action of a trajectory, $\sigma = \sigma(0..i)\sigma'$ where $\sigma' = \sigma(i+1..)$). These notions are similarly defined for traces and strings.

A trajectory (and naturally a trace or string) is *finite* iff the length $m \neq \infty$, and infinite otherwise. Runtime monitoring is supposed to monitor systems that run forever. This means monitoring infinite traces. In many cases it is useful to give results for systems that halt/fail/abort/deadlock. This means giving answers regarding finite traces. These finite traces are special in the sense that they should indicate somehow that the system has stopped unnatural and that nothing can happen after it stops. We follow Pratt [Pra79] and denote a special finite trace like this as ending in a special action denoted Λ. We call such traces *failing* traces (or trajectories). The *concatenation* of two trajectories σ' and σ'' is denoted $\sigma'\sigma''$ and is defined iff the trajectory σ' is finite and not failing and $\pi'(m) = \pi''(0)$ (i.e. the last state of σ' is the same as the first state of σ''). If σ' is failing then the concatenation $\sigma'\sigma''$ is just σ' as no observations can be done after a failing action. Concatenation is naturally extended to traces or strings; in the case of traces it is not the case to test the last and first actions, but in the case of string this is still required to test the last and first states.

The actions of a trace may be concurrent actions. We denote by $\alpha_\& \in \mathcal{A}_B \cup \mathcal{A}_B^\&$ actions which are either basic actions or concurrent actions (i.e.

compount actions generated from basic actions by using only the & operator). We denote by $\{\alpha_{\&}\} \subseteq \mathcal{A}_B$ the associated set of basic actions which form the concurrent action $\alpha_{\&}$. This is in accordance witht both the actions we defined for normative structures and for the guarded rooted trees. Henceforth we consider $\tau(i) = \{a, b, \ldots\}$ as a set of basic actions. Consequently we introduce an *enclosing* relation over traces defined as $\tau \supseteq \tau'$ iff $\forall i \in \mathbb{N}$, $\tau'(i) \subseteq \tau(i)$ where $\subseteq$ is the classical set inclusion. Note that this definition requires that $m_\tau \geq m_{\tau'}$. When we use the relation over trajectories $\sigma \supseteq \sigma'$ we understand it over the associated traces $\tau \supseteq \tau'$.

We define a *trajectory model* $M = (\sigma, \mathcal{V})$ corresponding to a trajectory σ as a pair formed of the trajectory together with a valuation function $\mathcal{V} : \mathcal{S} \rightarrow 2^{AP}$ which assigns to each state of the trajectory a set of asserions which are true in that state. We denote by $\mathcal{S}_\sigma$ the set of states of the trajectory σ (i.e. the codomain of π) and by $\mathcal{A}_\sigma$ the set of actions of trajectory σ (i.e. the codomain of τ). For brevity we write just $\mathcal{S}$ and $\mathcal{A}$ when σ is understood from the context. We denote by $\mathcal{M}_\sigma$ the set of all trajectory models associated to σ which are obtained by giving different valuation functions.

A trace $\tau \in T$ is said to be contained by the tree T iff τ represents a path in the tree (i.e. $\tau(0) = r$). Recall that we called a full path of a tree a path which ends in a leaf of the tree, and a subpath is one that does not end in a leaf. Naturally, any trace τ which is contained in a tree $I_{\mathcal{CA}}(\alpha)$ of an action α is finite as the trees are of finite depth (see Section 2.2.1). A trajectory is contained in a tree, denoted $\sigma \in T$ iff its associated trace is contained in the tree, i.e. $\tau \in T$. We consider here the set of all trajectories which are full paths in the tree T and denote it by $\|T\| = \{\sigma \mid \sigma \text{ a full path in } T\}$.

Recall the canonic form of an action $ACF^\alpha = +_{i \in I}\ \alpha^i_{\&} \cdot \alpha^i$ where I is an indexing set and $\alpha^i_{\&}$ can be basic actions a, concurrent actions $\alpha_{\&}$, or tests (therefore also $\mathbf{1} = \top?$ and $\mathbf{0}$). In other words $\alpha^i_{\&}$ is either a test or a set of basic actions. For the propositions below we consider negation only over actions that do not contain tests other than $\mathbf{1}$ and $\mathbf{0}$ the equivalent of $\top?$ and $\bot?$. Recall the negation of a compound action:

$$\overline{\alpha} = \overline{\underset{i \in I}{+} \alpha^i_{\&} \cdot \alpha^i} = \underset{\gamma \in \overline{R}}{+} \gamma \ + \ \underset{i \in I}{+} \alpha^i_{\&} \cdot \overline{\alpha^i}$$

where $\overline{R} = \{\gamma \,|\, \gamma \in \mathcal{A}_B^{\&} \text{ and } \forall i \in I, \{\alpha^i_{\&}\} \not\subseteq \{\gamma\}\}$.

Proposition 3.11 (Characterization of action negation with traces)**.**

The set of traces which are full paths of the tree interpreting the negation of $\alpha = +_{i \in I}\ \alpha^i_{\&} \cdot \alpha^i$ denoted by $\|I_{\mathcal{CA}}(\overline{\alpha})\|$ is equal to the set defined below.

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

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 $I_{\mathcal{CA}}(+_{\gamma\in\overline{R}}\gamma\ +\ +_{i\in I}\alpha^i_{\&}\cdot\overline{\alpha^i})$. The first part of the action, 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, i.e. $\sigma'(0)$, among those, but not all, 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. $+_{i\in I}\alpha^i_{\&}\cdot\overline{\alpha^i}$. 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 $\alpha^i=\mathbf{1}$ dissapear. This is because when negating $\overline{\alpha^i}=\overline{\mathbf{1}}=\mathbf{0}$ that 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 $\alpha^i_{\&}$ (i.e. $\alpha^i_{\&}=\sigma(0)$) and are followed by a trace (i.e. $\sigma(1..)$) which is part of the traces of the negation of the smaller compound action α^i. □

Note: We can use an equivalent characterization of the traces of length one where instead of testing for (non)inclusion of sets we can test for nonemptiness of set subtraction; i.e. replace $\{\alpha^i_{\&}\}\not\subseteq\tau(0)$ with $\{\alpha^i_{\&}\}\setminus\tau(0)\neq\emptyset$.

Proposition 3.12. *Any infinite trace τ of a trajectory is either starting with a trace bigger w.r.t. $\supseteq$ then a complete path of $I_{\mathcal{CA}}(\alpha)$ or it starts with trace bigger than a complete path of $I_{\mathcal{CA}}(\overline{\alpha})$.*

Proof: The proof is by *reductio ad absurdum*. If the trace $\tau\supseteq\tau^{I_{\mathcal{CA}}(\alpha)}\tau'$ starts with a full path of the tree $I_{\mathcal{CA}}(\alpha)$ the proof is finished. Suppose it is not the case that $\tau\supseteq\tau^{I_{\mathcal{CA}}(\alpha)}\tau'$. This means that $\exists i\leq h(I_{\mathcal{CA}}(\alpha))$ s.t. $\tau(0..i-1)\supseteq\tau^{I_{\mathcal{CA}}(\alpha)}(0..i-1)$ and for all ways $\tau^{I_{\mathcal{CA}}(\alpha)}(i)$ of extending the trace $\tau^{I_{\mathcal{CA}}(\alpha)}(0..i-1)$ in the tree $I_{\mathcal{CA}}(\alpha)$ it is the case that $\tau(i)\not\supseteq\tau^{I_{\mathcal{CA}}(\alpha)}(i)$. Consider the characterization of the negation of Proposition 3.11. It is easy to see that the trace $\tau^{I_{\mathcal{CA}}(\alpha)}(0..i-1)\tau(i)$ is a full path of the tree $I_{\mathcal{CA}}(\overline{\alpha})$ interpreting the negation of α because the first part is a trace of the action α and the last step of the trace respects the condition in the first set of states of Proposition 3.11. More explicitely, in Proposition 3.11 $\forall i\in I,\{\alpha^i_{\&}\}$ means

that *for all branches...* each action on the branch must not be less than the current element of the trace. This is the same as the argument needed above.

Considering that $\tau^{I_{\mathcal{CA}}(\alpha)}(0..i-1)\tau(i)$ is a full path of $I_{\mathcal{CA}}(\overline{\alpha})$ and that $\tau(0..i-1) \supseteq \tau^{I_{\mathcal{CA}}(\alpha)}(0..i-1)$ we finish the proof as the trace τ is starting with the trace $\tau(0..i) \supseteq \tau^{I_{\mathcal{CA}}(\overline{\alpha})}$ which is greater than a full path of the tree interpreting the negation of α. □

Giving semantics to the assertions or to the propositional and dynamic connectives of $\mathcal{CL}$ is rather standard. We define an entailment relation $\models$ over pairs $(\sigma, \mathcal{C})$ of traces and contracts, usually written $\sigma \models \mathcal{C}$ and we say that "trajectory σ *does not violate* the contract (clause) $\mathcal{C}$".

$\sigma \models \phi$ iff $\phi \in \mathcal{V}(\pi(0))$

$\sigma \models \mathcal{C}_1 \wedge \mathcal{C}_2$ iff $\sigma \models \mathcal{C}_1$ and $\sigma \models \mathcal{C}_2$

$\sigma \models \mathcal{C}_1 \vee \mathcal{C}_2$ iff $\sigma \models \mathcal{C}_1$ or $\sigma \models \mathcal{C}_2$

$\sigma \models \mathcal{C}_1 \oplus \mathcal{C}_2$ iff $(\sigma \models \mathcal{C}_1$ and $\sigma \not\models \mathcal{C}_2)$ or $(\sigma \not\models \mathcal{C}_1$ and $\sigma \models \mathcal{C}_2)$.

Because we are giving semantics in a linear model (i.e. a trajectory) the dynamic modality does not behave branching as it does in the semantics given in terms of normative structures. Moreover the two complementary [ ] and $\langle\ \rangle$ colapse to a single modality which for convenience we denote just $[\alpha]$ (this expresses "after doing action α").

$\sigma \models [\alpha]\mathcal{C}$ iff $\sigma' \in I_{\mathcal{CA}}(\overline{\alpha})$ a full path and $\sigma = \sigma'\sigma''$ or
$\sigma' \in I_{\mathcal{CA}}(\alpha)$ a full path and $\sigma \supseteq \sigma'\sigma''$ and $\sigma'' \models \mathcal{C}$

$\sigma \models \langle\alpha\rangle\mathcal{C}$ iff $\sigma \models [\alpha]\mathcal{C}$

The interesting part is the semantics for obligation $O_{\mathcal{C}}$ and prohibition $F_{\mathcal{C}}$ operators.

$\sigma \models O_{\mathcal{C}}(\alpha)$ iff $\exists\sigma' \in I_{\mathcal{CA}}(\alpha)$ a full path s.t. $\sigma \supseteq \sigma'\sigma''$
or $\sigma \models [\overline{\alpha}]\mathcal{C}$.

The obligation operator has two parts: one is the obligation itself $O(\alpha)$ of a compound action α (which may include choices, sequence, and concurrent execution at the basic actions level); the second part is the reparation $\mathcal{C}$ in case the obligation is violated ($O_{\mathcal{C}}(\alpha)$ is what in the deontic comunity is

termed as *contrary-to-duty*, i.e. CTD). Violation of an obligatory action is encoded by the action negation. The reparation $\mathcal{C}$ should be enforced *after* executing a trace which is part of the negation of the action α.

A trajectory σ respects an obligation $O_{\mathcal{C}}(\alpha)$ if either of the two complementary (see Proposition 3.12) conditions is satisfied. The first condition deals with the obligation itself. The trajectory σ respects the obligation $O(\alpha)$ if it starts with a finite trajectory s.t. each action in its trace includes the corresponding action of the trajectory σ' (this is what the operator $\supseteq$ is defined for earlier). Afterwords it may continue with any trajectory σ''. The trajectory σ' has to be a full path in the tree $I_{\mathcal{CA}}(\alpha)$ representing the action α. Note that it is required only the existsance of a full path σ' in the tree $I_{\mathcal{CA}}(\alpha)$. Obligation considers only a path of the tree and not the whole tree. This is because in a choice action $\alpha_1 + \alpha_2$ which is obligatory, respecting this choice obligation means that it must be executed one of the actions α_1 or α_2.

If this first condition is not respected then the only way of fulfilling the obligation is to respect the second condition. This simply says that the trajectory σ must satisfy $[\overline{\alpha}]\mathcal{C}$. See the proof of Proposition 3.12 for a more clear understanding of this behavior.

$$\begin{array}{l}\sigma \models F_{\mathcal{C}}(\alpha) \text{ iff } \forall \sigma' \in I_{\mathcal{CA}}(\alpha) \text{ a full path either} \\ \qquad\qquad\qquad\qquad \nexists \sigma'' \text{ s.t. } \sigma \supseteq \sigma'\sigma'' \qquad \text{or} \\ \qquad\qquad\qquad\qquad \sigma \supseteq \sigma'\sigma'' \text{ then } \sigma'' \models \mathcal{C}.\end{array}$$

An equivalent definition of the $\models$ relation which is also compositional is the following.

$$\begin{array}{l}\sigma \models [\alpha_{\&}]\mathcal{C} \text{ iff } \alpha_{\&} \subseteq \sigma(0) \text{ and } \sigma(1..) \models \mathcal{C} \text{ or} \\ \qquad\qquad\qquad\qquad \sigma(0) = \sigma'(0) \cup \sigma''(0) \wedge \sigma'(0) \subset \alpha_{\&} \wedge \sigma''(0) \subseteq \mathcal{A}_B^{\&} \backslash \alpha_{\&}.\end{array}$$

The second line in the definition above is the characterization of the tree defining the negation of only a concurrent action $\alpha_{\&}$. This characterization gives the same set of traces as the tree defined by Definition 2.13. The complete characterization of the action negation in terms of traces is given in Proposition 3.11.

A short note on the related PDL$^{\cap}$ with intersection of actions. In [BV03] interpretation of actions is modelled as the intersection of the associated relations.[17] The axiomatization presented there gives as asiom: $\langle \alpha \cap \beta \rangle \phi \Rightarrow \langle \alpha \rangle \phi$. One can infer from this the following theorem: $[\alpha]\phi \Rightarrow [\alpha \cap \beta]\phi$. In our linear models there is no more distinction between the existencial and universal flavours of the two dynamic modalities. On the other hand one can

[17] The same approach is taken in the other works in the literature on PDL$^{\cap}$ with intersection.

easily see that with the semantics given above if a trajectory σ satisfies $[\alpha_{\&}]\mathcal{C}$ then it will also satisfy $[\alpha_{\&}\&\beta_{\&}]\mathcal{C}$. This means that our semantics preserves the theorem given for PDL$^{\cap}$.

$\sigma \models [\alpha \cdot \beta]\mathcal{C}$ iff $\sigma \models [\alpha][\beta]\mathcal{C}$.

$\sigma \models [\alpha + \beta]\mathcal{C}$ iff $\sigma \models [\alpha]\mathcal{C}$ and $\sigma \models [\beta]\mathcal{C}$.

$\sigma \models [\beta^*]\mathcal{C}$ iff $\sigma \models \mathcal{C}$ and $\sigma \models [\beta][\beta^*]\mathcal{C}$.

$\sigma \models [\mathcal{C}_1?]\mathcal{C}_2$ iff $\sigma \not\models \mathcal{C}_1$ or $(\sigma \models \mathcal{C}_1$ and $\sigma \models \mathcal{C}_2)$.

Another theorem which holds in PDL$^{\cap}$ is $\langle\alpha\rangle\phi \wedge \langle\beta\rangle\varphi \Rightarrow \langle\alpha \cap \beta\rangle\phi \wedge \varphi$. We easily prove that a similar theorem holds in our semantics, which is natural considering the intuition:

$$[\alpha]\phi \wedge [\beta]\varphi \Rightarrow [\alpha\&\beta]\phi \wedge \varphi$$

$\sigma \models O_{\mathcal{C}}(\alpha_{\&})$ iff $\alpha_{\&} \subseteq \sigma(0)$ or $\sigma(1..) \models \mathcal{C}$.

$\sigma \models O_{\mathcal{C}}(\alpha \cdot \alpha')$ iff $\sigma \models O_{\mathcal{C}}(\alpha)$ and $\sigma \models [\alpha]O_{\mathcal{C}}(\alpha')$.

$\sigma \models O_{\mathcal{C}}(\alpha + \alpha')$ iff $\sigma \models O_{\bot}(\alpha)$ or $\sigma \models O_{\bot}(\alpha')$ or $\sigma \models [\overline{\alpha + \alpha'}]\mathcal{C}$.

The trace σ respects the obligation $O(\alpha_{\&})$ if the first action of the trace includes $\alpha_{\&}$. Otherwise, in case the obligation is violated,[18] the only way of fulfilling the contract is by respecting the reparation $\mathcal{C}$; i.e. $\sigma \models [\overline{\alpha_{\&}}]\mathcal{C}$. (See the proof of Proposition 3.12 in the appendix for a more clear understanding of this behavior.) Obligation considers only a path of the tree $I_{\mathcal{CA}}(\alpha)$ and not the whole tree. Therefore, 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.

$\sigma \models F_{\mathcal{C}}(\alpha_{\&})$ iff $\sigma(0) = \sigma'(0) \cup \sigma''(0) \wedge \sigma'(0) \subset \alpha_{\&} \wedge \sigma''(0) \subseteq \mathcal{A}_B^{\&} \setminus \alpha_{\&}$ or
$\alpha_{\&} \subseteq \sigma(0)$ and $\sigma(1..) \models \mathcal{C}$.

$\sigma \models F_{\mathcal{C}}(\alpha \cdot \alpha')$ iff $\sigma \models F_{\bot}(\alpha)$ or $\sigma \models [\alpha]F_{\mathcal{C}}(\alpha')$.

$\sigma \models F_{\mathcal{C}}(\alpha + \alpha')$ iff $\sigma \models F_{\mathcal{C}}(\alpha)$ and $\sigma \models F_{\mathcal{C}}(\alpha')$.

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

Recall that $O_\perp(\alpha)$ is denoted by just $O(\alpha)$ and is called cathegorical obligation which must not be violated. The other constructs of $\mathcal{CL}$ have the same semantics as given before.

Guided by the characterization of action negation in terms of traces of Proposition 3.11 it is easy to give $\sigma \models [\overline{\alpha + \beta}]\mathcal{C}$ in a compositional way so that we do not have to compute the negation of the action $\alpha + \beta$ which can be rather expensive.

$\sigma \models [\overline{\alpha_\&}]\mathcal{C}$ iff $\sigma(0) = \sigma'(0) \cup \sigma''(0) \wedge \sigma'(0) \subset \alpha_\& \wedge \sigma''(0) \subseteq \mathcal{A}_B^\& \setminus \alpha_\&$ and $\sigma(1..) \models \mathcal{C}$ or $\alpha_\& \subseteq \sigma(0)$.

$\sigma \models [\overline{\alpha \cdot \alpha'}]\mathcal{C}$ iff $\sigma \models [\overline{\alpha}]\mathcal{C}$ or $\sigma \models [\alpha][\overline{\alpha'}]\mathcal{C}$.

$\sigma \models [\overline{\alpha + \alpha'}]\mathcal{C}$ iff $\sigma \models [\overline{\alpha}]\mathcal{C}$ and $\sigma \models [\overline{\alpha'}]\mathcal{C}$.

Example 1. as traces: Consider the $\mathcal{CL}$ formula on page **??** which encodes the contract clause of the introduction. We give here few examples of traces of actions which *respect* the contract clause:

- ebl, p ("exceed bandwidth limit" and then "pay") which respects the contract because it respects the top level obligation;
- ebl, 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 ebl respects the contract.

Examples of traces which *violate* the clause are:

- ebl, ebl, ebl (constantly exceeding the limit) which violates both the first obligation and the second one by not paying;
- $ebl, d\&ne, d$ (after exceeding the bandwidth limit it constantly delays the payment) which again violates both obligations.

The relation $\models$ which we call "does not violate" is usually called in classical logic "models". This relation is extended to the "semantic entailment" relation $\Delta \models \mathcal{C}$ where Δ is a set of formulas of our $\mathcal{CL}$ logic. The above notation is read as "the formulas in Δ semantically entail $\mathcal{C}$". The semantic

entailment is defined as: for all trajectories which satisfy all the formulas of Δ they must also satisfy the formula $\mathcal{C}$ (i.e. $\forall\sigma$ s.t. $\sigma \models \mathcal{C}'$, $\forall \mathcal{C}' \in \Delta$ then $\sigma \models \mathcal{C}$). For some Δ, the set of all the formulas $\mathcal{C}$ s.t. $\Delta \models \mathcal{C}$ is called the *theory of* Δ and is denoted by $\mathbf{Th}^{\sigma}\Delta$. We denote by $\|\mathcal{C}\|^{\sigma} = \{\sigma \mid \sigma \models \mathcal{C}\}$ the set of all trajectories which are models of (i.e. satisfy) $\mathcal{C}$.

We say that a contract $\mathcal{C}'$ *respects* another contract $\mathcal{C}$ iff $\forall\sigma$ a run of $\mathcal{C}'$ then σ is also a run of $\mathcal{C}$ (e.g. for a contract $O(a\&b)$ all the runs respect the contract $O(a)$ and we can say that $O(a\&b)$ respects $O(a)$).

3.2.1 Properties of the linear semantics

Proposition 3.13 (properties on traces)**.**

$$O_{\top}(\alpha) \Leftrightarrow \top \tag{44}$$
$$O_{\mathcal{C}}(a) \wedge O_{\mathcal{C}}(b) \Leftrightarrow O_{\mathcal{C}}(a\&b) \tag{45}$$
$$O_{\mathcal{C}}(a) \not\models O_{\mathcal{C}}(a\&b) \tag{46}$$
$$O(a+b) \not\Leftrightarrow O(a\&b) \tag{47}$$
$$O(a+b) \not\Leftrightarrow O(a) \tag{48}$$
$$O(a+b) \not\Leftrightarrow O(a) \oplus O(b) \tag{49}$$
$$\textit{if } F(a) \textit{ then } F(a\&b) \tag{50}$$
$$F(a+b) \textit{ iff } F(a) \wedge F(b) \tag{51}$$
$$P(a+b) \textit{ iff } P(a) \wedge P(b) \tag{52}$$
$$F(a \cdot b) \textit{ iff } F(a) \vee \langle a\rangle F(b) \tag{53}$$
$$P(a \cdot b) \textit{ iff } P(a) \wedge [a]P(b) \tag{54}$$
$$\textit{is not the case that if } F(a\&b) \textit{ then } F(a) \tag{55}$$
$$\textit{is not the case that if } P(a\&b) \textit{ then } P(a) \tag{56}$$

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 obligation and make sure that they satisfy also the formula on the right.

For property (45): 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 $\tau(0) \supseteq a$ and $\tau(0) \supseteq b$. It implys that $\tau(0) \supseteq a\&b$ which means that $\sigma \models O(a\&b)$. The second implication $\Leftarrow$ is simpler and uses the same judgement. We leav that to the reader.

For property (46): This property is expressed in terms of the entailment relation and it says that it is not the case that $O_{\mathcal{C}}$ entails $O_{\mathcal{C}}(a\&b)$. Proving such kind of properties requires finding one model which satisfies the formula on the left and does not satisfy the formula on the right. From classical logic we know that $O_{\mathcal{C}}(a) \not\models O_{\mathcal{C}}(a) \wedge O_{\mathcal{C}}(a)$, and by property (45) the proof is finished. □

3.3 Relating the linear and the branching semantics

Theorem 3.14 (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 3.15. *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 [Koz81, 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 and Marcel Kyas 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.

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

[BV03] Philippe Balbiani and Dimiter Vakarelov. PDL with intersection of programs: A complete axiomatization. *Journal of Applied Non-Classical Logics*, 13(3-4):231–276, 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.

[CKS96] Ernie Cohen, Dexter Kozen, and Frederick Smith. Complexity of kleene algebra with tests, the. Technical report, Cornell University, Ithaca, NY, USA, 1996.

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

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

[Egg63] L. C. Eggan. Transition graphs and the star-height of regular events. *Michigan Mathematical Journal*, 10(4):385–397, 1963.

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

[FL77] Michael J. Fischer and Richard E. Ladner. Propositional modal logic of programs. In *9th ACM Symposium on Theory of Computing (STOC'77)*, pages 286–294. ACM, 1977.

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

[HTK00] David Harel, Jerzy Tiuryn, and Dexter Kozen. *Dynamic Logic*. MIT Press, Cambridge, MA, USA, 2000.

[Kap69] Donald M. Kaplan. Regular expressions and the equivalence of programs. *Journal of Computer and System Sciences*, 3(4):361–386, 1969.

[Koz81] Dexter Kozen. On the duality of dynamic algebras and kripke models. In *Logic of Programs, Workshop*, volume 125 of *Lecture Notes in Computer Science*, pages 1–11. Springer-Verlag, 1981.

[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.

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

[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.

[Pra86] Vaughan R. Pratt. Modeling concurrency with partial orders. *International Journal of Parallel Programming*, 15(1):33–71, Feb 1986.

[PS07a] Cristian Prisacariu and Gerardo Schneider. An algebraic structure for the Action-Based Contract language CL - Theoretical

Results. Technical Report 361, Department of Informatics, University of Oslo, July 2007.

[PS07b] Cristian Prisacariu and Gerardo Schneider. A formal language for electronic contracts. In *FMOODS'07*, volume 4468 of *LNCS*, pages 174–189. Springer, 2007.

[PS07c] 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.

[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.