UNIVERSITY OF OSLO
Department of Informatics

An Algebraic Structure for the Action-Based Contract Language $\mathcal{CL}$ – theoretical results[1]

Research Report No. 361

Cristian Prisacariu

Gerardo Schneider

Isbn 82-7368-319-2
Issn 0806-3036

July 2007

An Algebraic Structure for the Action-Based Contract Language $\mathcal{CL}$ – theoretical results[†]

Cristian Prisacariu[‡] Gerardo Schneider[§]

July 2007

Abstract

We introduce in this paper an algebra of actions specially tailored to serve as basis of an action-based formalism for writing electronic contracts. The proposed algebra is based on the work on Kleene algebras but does not consider the Kleene star and introduces a new constructor for modelling concurrent actions. The algebraic structure is resource-aware and incorporates special actions called tests. In order to be in accordance with the intuition behind electronic contracts we consider new properties of the algebraic structure, in particular a conflict relation and a demanding partial order. We also study a canonical form of the actions which, among other things, helps to naturally define a notion of action negation. Our action negation is more general than just negation of atomic actions, but more restricted than the negation involving the universal relation. A standard interpretation of the algebra is given in terms of guarded rooted trees with specially defined operations on them. The algebra is proven to be complete over the standard interpretation.

[†]Partially supported by the Nordunet3 project "Contract-Oriented Software Development for Internet Services".

[‡]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.

In [PS07a] 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 the 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, but we have left the formalization of the underlying action algebra underspecified.

In this paper we introduce a new algebraic structure to provide a well-founded formal basis for the action-based contract language $\mathcal{CL}$. Besides its use under the above-mentioned context, we believe the algebraic structure presented here is interesting by itself. 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 (see [PS07a]). A second difference is that we introduce an operator to model concurrency. The main contributions of the paper are: (1) A formalization of concurrent actions; (2) The introduction of a different kind of negation over actions; (3) A restricted notion of resource-awareness; and (4) A standard interpretation of the algebra over specially defined rooted trees. Among other, the interpretation using trees is intended to give in further work a particular semantics for the actions of the contract language of [PS07a].

The paper is organized as follows. The rest of the Introduction presents

briefly the $\mathcal{CL}$ contract language [PS07a]. In Section 2 we provide some algebraic background and useful terminology before introducing a first (core) version of the algebra for concurrent actions. In section 3 we give a standard interpretation of the algebra terms as rooted trees. The main result of this section is the completeness of the algebra over rooted trees interpretation. In section 4 we extend the algebra with boolean tests whereas in section 5 we introduce action negation and we discuss a normal form for our algebra. In Section 6 we introduce the reader to a more formal relation between the present algebra and the contract language $\mathcal{CL}$.[1] In the last section we conclude our work and give an extensive discussion on related works.

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

In this section we recall the contract language $\mathcal{CL}$; for a more detailed presentation see [PS07a].

Definition 1.1 (Contract Language Syntax). *A contract is defined by:*

$$\begin{array}{rcl} \mathcal{C}ontract & := & \mathcal{D}\ ;\ \mathcal{C} \\ \mathcal{C} & := & \phi \mid \mathcal{C}_O \mid \mathcal{C}_P \mid \mathcal{C}_F \mid \mathcal{C} \wedge \mathcal{C} \mid [\alpha]\mathcal{C} \mid \langle\alpha\rangle\mathcal{C} \mid \mathcal{C}\,\mathcal{U}\,\mathcal{C} \mid \bigcirc\mathcal{C} \mid \Box\mathcal{C} \\ \mathcal{C}_O & := & O(\alpha) \mid \mathcal{C}_O \oplus \mathcal{C}_O \\ \mathcal{C}_P & := & P(\alpha) \mid \mathcal{C}_P \oplus \mathcal{C}_P \\ \mathcal{C}_F & := & F(\alpha) \mid \mathcal{C}_F \vee [\alpha]\mathcal{C}_F \end{array}$$

The syntax of $\mathcal{CL}$ closely resembles the syntax of a modal (deontic) logic. Though this similarity is clearly intentional since we are driven by a logic-based approach, $\mathcal{CL}$ is *not* a logic. The interpretation of the $\mathcal{CL}$ syntax is given by translating it into an extension of μ-calculus which we call $\mathcal{C}\mu$. In what follows we provide an intuitive explanation of the $\mathcal{CL}$ syntax.

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 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.

[1] An extensive investigation in this direction is carried out in a subsequent paper.

$\mathcal{C}$ is the general *contract clause*. $\mathcal{C}_O$, $\mathcal{C}_P$, and $\mathcal{C}_F$ denote respectively *obligation*, *permission*, and *prohibition* clauses. $O(\cdot)$, $P(\cdot)$, and $F(\cdot)$, represents the obligation, permission or prohibition of performing a given action. $\wedge$ 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. α is a compound action (i.e., an expression containing one or more of the following operators: choice "+"; sequence "$\cdot$"; concurrency "&", and test "?" —see [PS07b]). Note that syntactically $\oplus$ cannot appear between prohibitions.

We borrow from propositional dynamic logic [FL77] the syntax $[\alpha]\phi$ to represent that after performing α (if it is possible to do so), ϕ must hold. The $[\cdot]$ notation allows having a *test*, where $[\phi?]\mathcal{C}$ must be understood as $\phi \Rightarrow \mathcal{C}$. $\langle\alpha\rangle\phi$ captures the idea that it exists the possibility of executing α, in which case ϕ must hold afterwards. Following temporal logic (TL) notation we have $\mathcal{U}$ (*until*), $\bigcirc$ (*next*), and $\Box$ (*always*), with intuitive semantics as in TL [Pnu77]. Thus $\mathcal{C}_1 \mathcal{U} \mathcal{C}_2$ states that $\mathcal{C}_1$ holds until $\mathcal{C}_2$ holds. $\bigcirc\mathcal{C}$ intuitively states that $\mathcal{C}$ holds in the next moment, usually after something happens, and $\Box\mathcal{C}$ expressing that $\mathcal{C}$ holds in every moment. We can define $\Diamond\mathcal{C}$ (*eventually*) for expressing that $\mathcal{C}$ holds sometimes in a future moment.

To express CTDs we provide the following notation, $O_\varphi(\alpha)$, which is syntactic sugar for $O(\alpha) \wedge [\overline{\alpha}]\varphi$ stating the obligation to execute α, and the reparation φ in case the obligation is violated, i.e. whenever α is not performed. The reparation may be any contract clause which is formed only of O and F expressions. Similarly, CTP statements $F_\varphi(\alpha)$ can be defined as $F_\varphi(\alpha) = F(\alpha) \wedge [\alpha]\varphi$, where φ is the penalty in case the prohibition is violated. Notice that it is possible to express nested CTDs and CTPs.

In $\mathcal{CL}$ we can write *conditional* obligations, permissions and prohibitions in two different ways. Just as an example let us consider conditional obligations. The first kind is represented as $[\alpha]O(\beta)$, which may be read as "after performing α, one is obliged to do β". 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 $[\phi?]\mathcal{C}$.

2 Algebra of Concurrent Actions

2.1 Background

We recall that a Kleene algebra is a structure $\mathcal{K} = (\mathbf{K}, +, \cdot, \mathbf{0}, \mathbf{1}, ^*)$ with the properties that $(\mathbf{K}, +, \mathbf{0})$ is a commutative monoid with the identity element $\mathbf{0}$, and $(\mathbf{K}, \cdot, \mathbf{1})$ is a monoid with the identity element $\mathbf{1}$. Moreover, the operator $+$ is idempotent and thus the structure $(\mathbf{K}, +, \cdot, \mathbf{0}, \mathbf{1})$ is an idempotent semiring. The * is a unary operator with the intuition that $a^* = 1 + a + a \cdot a + \ldots$ (e.g. if the elements of $\mathbf{K}$ are considered as relations over a set X, and $\mathbf{1}$, $+$, and $\cdot$ are the usual identity relation, relation union, and relation composition respectively then a^* is the transitive reflexive closure of relation a). A nice axiomatization of * was given in [Con71]. In programming theory it is usual to interpret $+$ as *choice*, $\cdot$ as *sequence* and * as *iteration*.

$(\mathbf{K}, +, \mathbf{0})$ being a commutative monoid means that the following should hold:

$$x + (y + z) = (x + y) + z \tag{1}$$
$$x + y = y + x \tag{2}$$
$$\mathbf{0} + x = x + \mathbf{0} = x \tag{3}$$

Equations (1), (2), and (3) define respectively the *associativity*, the *commutativity*, and the *identity element* properties of the commutative monoid.

The $+$ is defined to respect the following *idempotent* equivalence:

$$x + x = x \tag{4}$$

For the monoid $(\mathbf{K}, \cdot, \mathbf{1})$ we do not have commutativity; i.e. we have as axioms only the following:

$$x \cdot (y \cdot z) = (x \cdot y) \cdot z \tag{5}$$
$$\mathbf{1} \cdot x = x \cdot \mathbf{1} = x \tag{6}$$

We note that $\mathbf{0}$ is an *annihilator* of $\cdot$ operator:

$$\mathbf{0} \cdot x = x \cdot \mathbf{0} = \mathbf{0} \tag{7}$$

Moreover, $\cdot$ is defined to be distributive over $+$ both on the left and on the right:

$$x \cdot (y + z) = x \cdot y + x \cdot z \tag{8}$$
$$(x + y) \cdot z = x \cdot z + y \cdot z \tag{9}$$

The two monoid structures above with the properties that we have seen given by axioms (7), (8), and (9) form a structure $(\mathbf{K}, +, \cdot, \mathbf{0}, \mathbf{1})$ which is called a *semiring*. A semiring is called *idempotent* if the $+$ operator respects the idempotence equivalence (4). A *Kleene algebra with tests* is a rather more complex structure $\mathcal{D} = (\mathcal{K}, \mathcal{B})$ where $\mathcal{K}$ is a Kleene algebra and $\mathcal{B}$ is a classical *Boolean algebra*. The elements of the Boolean algebra are called test and are included in the set of elements of the Kleene algebra $\mathcal{K}$.

2.2 The algebraic structure $\mathcal{CA}$

We start by defining an algebraic structure $\mathcal{CA} = (\mathcal{A}, +, \cdot, \&, \mathbf{0}, \mathbf{1})$ which is the basis of the *algebra of concurrent actions and tests* that we put forward in this paper. $\mathcal{CA}$ defines the concurrent actions and the Boolean algebra presented in Section 4 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 (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 $\mathcal{A}_B$ is called the *generator set* of the algebra. The basic actions of $\mathcal{A}_B$ 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 $\mathcal{A}_B$. We will just use $T_{\mathcal{CA}}$ whenever $\mathcal{A}_B$ is understood by context.

Definition 2.1 (action terms).

1. *any basic action a of $\mathcal{A}_B$ is an action of $\mathcal{A}$;*

(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)	$\alpha\&(\beta + \gamma) = \alpha\&\beta + \alpha\&\gamma$
(6)	$\alpha \cdot \mathbf{1} = \mathbf{1} \cdot \alpha = \alpha$	(15)	$(\alpha + \beta)\&\gamma = \alpha\&\gamma + \beta\&\gamma$
(7)	$\alpha \cdot \mathbf{0} = \mathbf{0} \cdot \alpha = \mathbf{0}$	(16)	$\alpha\&(\alpha' \cdot \beta) = \alpha(1)\&\alpha'(1) \cdot \ldots \cdot \alpha(n)\&\alpha'(n) \cdot \beta$
(8)	$\alpha \cdot (\beta + \gamma) = \alpha \cdot \beta + \alpha \cdot \gamma$		where $l(\alpha) = l(\alpha') = n$
(9)	$(\alpha + \beta) \cdot \gamma = \alpha \cdot \gamma + \beta \cdot \gamma$		

Table 1: Axioms of $\mathcal{CA}$

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* $\mathcal{A}$*.*

Throughout this paper 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 neutral elements for $+$, respectively for $\cdot$ and $\&$ operators. We call action $\mathbf{1}$ the *skip* action. In Table 1 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 1. Axioms (1)-(4) define $+$ to be associative, commutative, with neutral element $\mathbf{0}$, and idempotent. Axioms (5)-(7) define $\cdot$ to be associative, with neutral element $\mathbf{1}$, and with annihilator $\mathbf{0}$. The element $\mathbf{0}$ is an annihilator for the sequence operator both on the left and 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 5 when we define a *canonical 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 *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 such semantics in this paper. 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 $\mathcal{A}_{\&}$ the subset of elements of $\mathcal{A}$ generated using only & constructor (e.g. a, $a\&a, a\&b \in \mathcal{A}_{\&}$ and $a + b, a\&b + c, a \cdot b \notin \mathcal{A}_{\&}$).

Axioms (10)-(13) give the properties of & to be associative, commutative, with neutral element $\mathbf{1}$, and annihilator $\mathbf{0}$ 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 +). Axioms (14) and (15) define the distributivity of & over +. From axioms (10)-(15) 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 semiring.

For axiom (16) we need some preliminary notions introduced in the following. We consider that basic actions are instantaneous 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. $l(a) = 1$ *for any basic action* a *of* $\mathcal{A}_B$,
2. $l(\alpha\&\beta) = l(\alpha + \beta) = max(l(\alpha), l(\beta))$,
3. $l(\alpha \cdot \beta) = l(\alpha) + l(\beta)$.

$max : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is the standard function returning the maximum value of the two arguments. For the special action $\mathbf{1}$ the length is $l(\mathbf{1}) = 0$. 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 $\alpha = (a + b) \cdot c$ we have $l(\alpha) = 2$, $\alpha(1) = a + b$ and $\alpha(2) = c$.

Specific to our application domain we consider it is natural to relate & and $\cdot$ as follows. The equation is based on some properties of the actions lengths.

$$\text{if } l(\alpha)=l(\alpha')=n \text{ then } \alpha\&(\alpha'\cdot\beta) = \alpha(1)\&\alpha'(1)\cdot\ldots\cdot\alpha(n)\&\alpha'(n)\cdot\beta \quad (16)$$

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. Note that $\mathbf{1}$ is ignored in the equation above 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 $a\&b$.

Let us take a look at the properties of $+$ and $\&$ of being respectively idempotent and not idempotent; for $+$ we have $\alpha+\alpha=\alpha$, but for $\&$ the equation does not hold. 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 $\mathcal{A}_B$.

On the other hand, we want to have a resource-aware algebra similarly to what has been done for linear logic [Gir87]. For this we do not allow the idempotence property for the $\&$ operator ($a\&a\neq a$). As an example, if a represents the action of paying 100\$ then paying 200\$ would be represented as $a\&a$. Note that we can represent only discrete quantities with this approach. Therefore, for concurrent actions of $\mathcal{A}_\&$ we may have any number of duplicate atomic actions in its composition; i.e. $a\&a\&b$ and $a\&b$ are different therefore, the basic actions of a concurrent action α form a *multiset* over $\mathcal{A}_B$.

We recall here that the notion of a multiset M over a set A is a function $M:A\to\mathbb{N}$, where intuitively $M(a)$ is the number of copies of element $a\in A$. Informally, a multiset is a set where the number of occurrences of an element does matter.

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 multisets 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 multiset is equivalent to a pomset with the empty order as the partial order. 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 multiset are executed concurrently. Note that in the same theory of pomsets a set is also a pomset with the empty order (and an additional condition of injective labelling). The reason for which we consider multisets and not just sets is that we want to model concurrent execution of several copies of the same action.

With the view of concurrent actions as multisets over $\mathcal{A}_B$ we can define a strict partial order over concurrent actions with the help of inclusion of multisets. We recall that $M \subset N$ iff $\forall a \in A$ we have $M(a) \leq N(a)$ and $\exists a \in A$ such that $M(a) < N(a)$, which says that M is included in N if and only if we can remove from N each element of M and not get the empty multiset.

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

$$\alpha <_{\&} \beta \stackrel{def}{=} M_\alpha \subset M_\beta \tag{17}$$

where α and β are concurrent actions, and M_α denotes the multiset associated to α.

We call $<_{\&}$ the *demanding relation* with the intuition that β is more demanding than α. We consider the action $\mathbf{1}$ as the empty multiset, 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 $<_{\&}$.

Proposition 2.1. *The relation $<_{\&}$ is a strict partial order.*

Proof: It is easy to prove that $<_{\&}$ is a strict partial order:

1. ireflexivity: $\nexists \alpha$ s.t. $\alpha <_{\&} \alpha$ because $M_\alpha \not\subset M_\alpha$ as $\forall a \in \alpha$, $M_\alpha(a) \leq M_\alpha(a)$ so the second part of the definition of $\subset$ is not respected;

2. transitivity: if $\alpha <_{\&} \beta$ then $M_\alpha \subset M_\beta$, and if $\beta <_{\&} \gamma$ then $M_\beta \subset M_\gamma$. By the transitivity of the inclusion relation of multisets we get our result of transitivity;

3. antisymmetry: $\forall \alpha \neq \beta$ if $\alpha <_{\&} \beta$ then $M_\alpha \subset M_\beta$ which means that $M_\beta \not\subset M_\alpha$ so $\beta \not<_{\&} \alpha$. Note that antisymmetry is not required as it follows from ireflexivity and transitivity.

□

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 multisets for the & to some more complex definitions for $\cdot$ and $+$.

As we have seen the length function considers actions as strings. Henceforth we consider actions as *strings of multisets*. As an example, take the concurrent compound actions α, β, and γ with the associated multisets M_α, M_β, and M_γ, respectively, then the action $\alpha \cdot \beta \cdot \gamma$ has associated the following string of multisets $\langle M_\alpha M_\beta M_\gamma \rangle$. With this representation one can give several ways of extending the $<_{\&}$ to the sequence actions; we take the most natural one: two sequence actions $\alpha_1 \cdot \ldots \cdot \alpha_n$ and $\beta_1 \cdot \ldots \cdot \beta_m$ are comparable by the $<_{\&}$ order iff their associated strings of multisets $\langle M_{\alpha_1} \ldots M_{\alpha_n} \rangle$ and $\langle M_{\beta_1} \ldots M_{\beta_m} \rangle$ can be compared as follows:

Without loss of generality, consider $n \leq m$ with $n, m \in \mathbb{N}$ then

1. if $\exists 1 \leq i \leq n$ s.t. $\forall j \leq i$ $M_{\alpha_j} \not\subset M_{\beta_j}$ and $M_{\beta_j} \not\subset M_{\alpha_j}$ and
   (a) $M_{\alpha_i} \subset M_{\beta_i}$ then $\alpha_1 \cdot \ldots \cdot \alpha_n <_{\&} \beta_1 \cdot \ldots \cdot \beta_m$;
   (b) $M_{\beta_i} \subset M_{\alpha_i}$ then $\beta_1 \cdot \ldots \cdot \beta_m <_{\&} \alpha_1 \cdot \ldots \cdot \alpha_n$;
2. if $\forall i \leq n$, $M_{\alpha_i} \not\subset M_{\beta_i}$ and $M_{\beta_i} \not\subset M_{\alpha_i}$
   then $\alpha_1 \cdot \ldots \cdot \alpha_n \not<_{\&} \beta_1 \cdot \ldots \cdot \beta_m$ and $\beta_1 \cdot \ldots \cdot \beta_m \not<_{\&} \alpha_1 \cdot \ldots \cdot \alpha_n$.

Intuitively for sequence actions the $<_{\&}$ order starts to compare from left to right each element of the sequence and it stops at the first comparable (by $<_{\&}$ on concurrent actions) pair and returns the corresponding result. Consider the following examples: $a \cdot b <_{\&} a \cdot b\&c$; $a \cdot b\&c <_{\&} b \cdot a\&b\&c$; $a \cdot a\&b \not<_{\&} b \cdot a\&c$. Moreover, because $\mathbf{1} <_{\&} a$, $\forall a \in \mathcal{A}_B$ and $\mathbf{1}$ can be appended to any action without changing it then $\alpha <_{\&} \alpha \cdot \beta$ as $\alpha \cdot \mathbf{1} <_{\&} \alpha \cdot \beta$ enters under the case 1. above.

As we have seen, we can associate sets to choice actions and therefore, if we consider choice only of concurrent actions then for an action $\alpha = \alpha_1 + \ldots + \alpha_n$ with $\alpha_i = a_1\& \ldots \&a_m$ we associate a *set of multisets* $\{M_{\alpha_1}, \ldots, M_{\alpha_n}\}^\alpha$ where the superscript is the name of the action α. We first give the extension of $<_{\&}$ to this kind of compound actions. Take two actions constructed as

above, $\alpha = \alpha_1 + \ldots + \alpha_n$ and $\beta = \beta_1 + \ldots + \beta_m$ with their associated sets of multisets $\{M_{\alpha_1}, \ldots, M_{\alpha_n}\}^\alpha$ and $\{M_{\beta_1}, \ldots, M_{\beta_m}\}^\beta$. We say that $\alpha <_\& \beta$ iff there exists a function $f : \{\ldots\}^\beta \to \{\ldots\}^\alpha$ defined on the set of multisets given by action β with the results in the set of multisets given by α such that $f(M_{\beta_i}) = M_{\alpha_j}$ iff $M_{\alpha_j} \subset M_{\beta_i}$. Otherwise we say that $\alpha \not<_\& \beta$.

For more complex choice actions where we have sequence of actions instead of just concurrent actions the definition is similar just that we do not have sets of multisets, but *sets of strings of multisets* and instead of using in the definition of the function f the inclusion of multisets we use $<_\&$ defined for strings of multisets.

We can now give the following result.

Theorem 2.2. *The operators $\&$, $\cdot$, and $+$ are monotone with respect to the demanding relation; i.e. for the relation $<_\&$ and for any actions α, β, and γ we have:*

$$\begin{array}{lll} \textit{if } \alpha <_\& \beta & \textit{then} & \alpha\&\gamma <_\& \beta\&\gamma \\ \textit{if } \alpha <_\& \beta & \textit{then} & \alpha \cdot \gamma <_\& \beta \cdot \gamma \quad \textit{and} \quad \gamma \cdot \alpha <_\& \gamma \cdot \beta \\ \textit{if } \alpha <_\& \beta & \textit{then} & \alpha + \gamma <_\& \beta + \gamma \end{array}$$

Proof: The actions α, β, and γ can be any compound action of $\mathcal{CA}$. Therefore, the proof should take into account all the forms of an action. A much simpler and clear proof can be given after we define the canonical form of an action in Section 5. We only take here a few simple cases to illustrate the proof procedure. The other cases are treated similarly.

Case 1 (operator $\&$).

Let us take first a simple example: if $a <_\& a\&b$ then $a\&c <_\& a\&b\&c$.

For this case we consider only concurrent compound actions. For the actions α, β, and γ we have associated the multisets M_α, M_β, and M_γ. Note that operator $\&$ relates to the union of multisets; i.e. for action $\alpha\&\gamma$ we have associated the multiset $M_\alpha \cup M_\gamma$. It is now simple to see that if $\alpha <_\& \beta$ then $M_\alpha \subset M_\beta$ which implies $M_\alpha \cup M_\gamma \subset M_\beta \cup M_\gamma$ which gives $\alpha\&\gamma <_\& \beta\&\gamma$.

Case 2 (operator $\cdot$).

An example is: for $a <_\& a\&b$ we want $a \cdot c <_\& a\&b \cdot c$ (recall that we have the precedence $\&, >, \cdot$ to avoid using parenthesis).

For this case we consider that the actions are sequences of concurrent actions. The actions α, β, and γ have associated the strings of multisets $\langle M_{\alpha_1} \ldots M_{\alpha_n} \rangle$, $\langle M_{\beta_1} \ldots M_{\beta_m} \rangle$, and $\langle M_{\gamma_1} \ldots M_{\gamma_k} \rangle$. Sequence of two actions $\alpha \cdot \gamma$ is concatenation of the corresponding strings of multisets $\langle M_{\alpha_1} \ldots M_{\alpha_n} M_{\gamma_1} \ldots M_{\gamma_k} \rangle$.

We have that $\alpha <_{\&} \beta$ which by definition means that $\exists i \leq n$ such that $\forall j \leq i$ $M_{\alpha_j} \not\subset M_{\beta_j}$ and $M_{\beta_j} \not\subset M_{\alpha_j}$ and $M_{\alpha_i} \subset M_{\beta_i}$. These facts hold also after concatenation of strings of multisets; i.e. for strings $\langle M_{\alpha_1} \ldots M_{\alpha_n} M_{\gamma_1} \ldots M_{\gamma_k} \rangle$ and $\langle M_{\beta_1} \ldots M_{\beta_m} M_{\gamma_1} \ldots M_{\gamma_k} \rangle$. We have thus the definition satisfied for the sequence actions, i.e. $\alpha \cdot \gamma <_{\&} \beta \cdot \gamma$.

Case 3 (operator +).

An example is: if $a <_{\&} a\&b$ we have $a + c <_{\&} a\&b + c$.

For this case we take choice of concurrent actions. The actions α, β, and γ have associated the sets of multisets $\{M_{\alpha_1}, \ldots, M_{\alpha_n}\}^{\alpha}$, $\{M_{\beta_1}, \ldots, M_{\beta_m}\}^{\beta}$, and $\{M_{\gamma_1}, \ldots, M_{\gamma_k}\}^{\gamma}$. The choice + on actions relates to the union of sets of multisets. For example, for action $\alpha + \gamma$ we have the set $\{M_{\alpha_1}, \ldots, M_{\alpha_n}\}^{\alpha} \cup \{M_{\gamma_1}, \ldots, M_{\gamma_k}\}^{\gamma} = \{M_{\alpha_1}, \ldots, M_{\alpha_n}, M_{\gamma_1}, \ldots, M_{\gamma_k}\}$. By the hypothesis we have $\alpha <_{\&} \beta$ which by the definition means that there exists a function $f : \{M_{\beta_1}, \ldots, M_{\beta_m}\} \to \{M_{\alpha_1}, \ldots, M_{\alpha_n}\}$ such that $\forall 1 \leq i \leq m$ $\exists 1 \leq j \leq n$ such that $M_{\alpha_j} \subset M_{\beta_i}$.

To prove that $\alpha + \gamma <_{\&} \beta + \gamma$ we have to find a function $f' : \{\}^{\beta+\gamma} \to \{\}^{\alpha+\gamma}$ such that $f'(M_{\beta+\gamma_j}) = M_{\alpha+\gamma_i}$ means that $M_{\alpha+\gamma_i} \subset M_{\beta+\gamma_j}$. We define f' as an extension of f where $f'(M_{\beta_j}) = f(M_{\beta_j})$ and $f'(M_{\gamma_j}) = M_{\gamma_j}$. The definition of f' respects the definition and thus we have that $\alpha + \gamma <_{\&} \beta + \gamma$.

For choice of sequence actions the proof is similar just that it takes $<_{\&}$ instead of the inclusion of multisets. □

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. The *preference relation* is defined as:

$$\alpha \geq \beta \iff^{def} \alpha + \beta = \alpha \tag{18}$$

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.

For the preference relation to be a partial order we prove that the following properties hold:

1. reflexivity: for all α we have $\alpha \geq \alpha$;

   Proof: The proof is immediate from the idempotence property of +. $\alpha \geq \alpha \iff^{def} \alpha + \alpha = \alpha$ which is true from equation (4). □

2. transitivity: for all α, β, γ if $\alpha \geq \beta$ and $\beta \geq \gamma$ then $\alpha \geq \gamma$;

Proof: From $\alpha \geq \beta$ and $\beta \geq \gamma$ we have to prove $\alpha + \gamma = \alpha$. We have $\alpha + \beta = \alpha$ then $\alpha + \beta + \gamma = \alpha + \gamma$ from the associativity of $+$. Together with $\beta + \gamma = \beta$ we get $\alpha + \beta = \alpha + \gamma$. Using again the first equation we get that $\alpha = \alpha + \gamma$ which is what we wanted. □

3. antisymmetry: for all α, β, if $\alpha \geq \beta$ and $\beta \geq \alpha$ then $\alpha = \beta$ (α and β are the same element).

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

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.

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 2.4 (conflict and compatibility).
The conflict relation is defined as:

$$a \#_{\mathcal{C}} b \stackrel{def}{\iff} a\&b = \mathbf{0} \tag{19}$$

The compatibility relation is defined as:

$$a \sim_{\mathcal{C}} b \stackrel{def}{\iff} a\&b \neq \mathbf{0}, \text{ where } a, b \neq \mathbf{0} \tag{20}$$

Proposition 2.3. *The following standard properties of the conflict and compatibility relations for basic actions hold:*

1. *reflexivity of* $\sim_{\mathcal{C}}$*:* $a \sim_{\mathcal{C}} a$*. Any basic action is compatible with itself;*

   Proof: *if* $a \neq \mathbf{0}$ *then* $a\&a \neq \mathbf{0}$ *then, by definition* $a \sim_{\mathcal{C}} a$. □

2. *symmetry of* $\#_{\mathcal{C}}$ *or* $\sim_{\mathcal{C}}$*:* *if* $a \#_{\mathcal{C}} b$ *then* $b \#_{\mathcal{C}} a$*, and if* $a \sim_{\mathcal{C}} b$ *then* $b \sim_{\mathcal{C}} a$*. There is no order on the actions that are in conflict or compatible.*

   Proof: *The proof follows immediately from the symmetry of* &*. For* $a \#_{\mathcal{C}} b$ *then* $a\&b = \mathbf{0}$ *which means that* $b\&a = \mathbf{0}$ *which is* $b \#_{\mathcal{C}} a$. □

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.

Proposition 2.4 (extension of $\#_{\mathcal{C}}$ to compound actions).

1. *$\alpha \#_{\mathcal{C}} \beta \Rightarrow \alpha \#_{\mathcal{C}} \gamma$, $\forall \gamma$ a concurrent action (i.e. constructed from basic actions only with $\&$) s.t. $\beta <_{\&} \gamma$.*

   Proof: *$\alpha \#_{\mathcal{C}} \beta \stackrel{def}{\Rightarrow} \alpha\&\beta = \mathbf{0}$ which $\forall \beta'$ $\alpha\&\beta\&\beta' = \mathbf{0}\&\beta' = \mathbf{0}$ which means that $\alpha \#_{\mathcal{C}} \beta\&\beta'$, i.e. $\alpha \#_{\mathcal{C}} \gamma$ and $\beta <_{\&} \gamma = \beta\&\beta'$.*

   Ex.: $a \#_{\mathcal{C}} b$ then $a \#_{\mathcal{C}} b\&b$ and $a \#_{\mathcal{C}} b\&c$. □

2. *$\alpha \#_{\mathcal{C}} \beta$ and $\alpha \#_{\mathcal{C}} \gamma$ then $\alpha \#_{\mathcal{C}} \beta + \gamma$.*

   Proof: *$\alpha \#_{\mathcal{C}} \beta \Rightarrow \alpha\&\beta = \mathbf{0}$ and $\alpha \#_{\mathcal{C}} \gamma \Rightarrow \alpha\&\gamma = \mathbf{0}$. From these we have that $(\alpha\&\beta) + (\alpha\&\gamma) = \mathbf{0} + \mathbf{0} = \mathbf{0} \stackrel{(14)}{\Rightarrow} \alpha\&(\beta + \gamma) = \mathbf{0}$ which by definition means that $\alpha \#_{\mathcal{C}} \beta + \gamma$.* □

3. *$\alpha \#_{\mathcal{C}} \beta \Rightarrow \alpha \#_{\mathcal{C}} \beta \cdot \gamma$, $\forall \gamma \in \mathcal{A}$.*

   Proof: *$\alpha \#_{\mathcal{C}} \beta \Rightarrow \alpha\&\beta = \mathbf{0}$ which means that $(\alpha\&\beta) \cdot \gamma = \mathbf{0} \stackrel{(16)}{\Rightarrow} \alpha\&(\beta \cdot \gamma) = \mathbf{0}$ which by definition $\alpha \#_{\mathcal{C}} \beta \cdot \gamma$.* □

Proposition 2.5 (extension of $\sim_{\mathcal{C}}$ to compound actions).

1. *$\alpha \sim_{\mathcal{C}} \alpha \cdot \gamma$, $\forall \gamma \neq \mathbf{0}$.*

   Proof: *By reflexivity of $\sim_{\mathcal{C}}$ we have that $\alpha \sim_{\mathcal{C}} \alpha \Rightarrow \alpha\&\alpha \neq \mathbf{0}$ which (because $\gamma \neq \mathbf{0}$) means that $(\alpha\&\alpha) \cdot \gamma \neq \mathbf{0} \stackrel{(16)}{\Rightarrow} \alpha\&(\alpha \cdot \gamma) \neq \mathbf{0}$ and thus $\alpha \sim_{\mathcal{C}} \alpha \cdot \gamma$.* □

2. *$\alpha \sim_{\mathcal{C}} \beta \Rightarrow \alpha \sim_{\mathcal{C}} \beta \cdot \gamma$, $\forall \gamma \neq \mathbf{0}$.*

   Proof: *$\alpha \sim_{\mathcal{C}} \beta \Rightarrow \alpha\&\beta \neq \mathbf{0}$ which means (because $\gamma \neq \mathbf{0}$) that $(\alpha\&\beta) \cdot \gamma \neq \mathbf{0} \stackrel{(16)}{\Rightarrow} \alpha\&(\beta \cdot \gamma) \neq \mathbf{0}$ and thus $\alpha \sim_{\mathcal{C}} \beta \cdot \gamma$.* □

3. *$\alpha \sim_{\mathcal{C}} \beta \Rightarrow \alpha \sim_{\mathcal{C}} \alpha + \beta$ and $\beta \sim_{\mathcal{C}} \alpha + \beta$.*

   Proof: *We prove only the first part as the second implication is similar. $\alpha \sim_{\mathcal{C}} \beta \Rightarrow \alpha\&\beta \neq \mathbf{0}$ which means that $(\alpha\&\beta) + (\alpha\&\alpha) \neq \mathbf{0}$ as it*

does not matter what is the second argument of the + as long as the first argument is different from $\mathbf{0}$. *From (14) we get* $\alpha\&(\beta+\alpha) \neq \mathbf{0}$ *which by definition is* $\alpha \sim_{\mathcal{C}} \alpha+\beta$. □

4. $\alpha \sim_{\mathcal{C}} \beta \Rightarrow \alpha \sim_{\mathcal{C}} \alpha\&\beta$ *and* $\beta \sim_{\mathcal{C}} \alpha\&\beta$.

   Proof: $\alpha \sim_{\mathcal{C}} \beta \Rightarrow \alpha\&\beta \neq \mathbf{0}$ *together with reflexivity of* $\sim_{\mathcal{C}}$ *we have* $\alpha\&\beta\&\alpha \neq \mathbf{0}$ *which by definition* $\alpha \sim_{\mathcal{C}} \alpha\beta$. □

3 Standard Interpretation using 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.

3.1 Rooted trees

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

Definition 3.1 (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[2] 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*[3] 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 respect to the partial order. Note that there cannot be two edges (n, m) and

[2] 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.

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

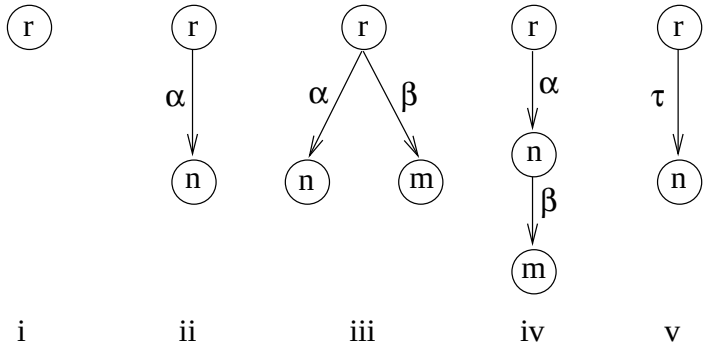

Figure 1: Examples of finite rooted trees with labeled edges.

(m, n) in the same tree. All nodes $\{m \,|\, (n, m)\}$ are called the descendents (or children) nodes of n.

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 α are multisets of *basic labels* of A; e.g. $\alpha_1 = \{a, b\}$ or $\alpha_2 = \{a, a\}$, or $\alpha_2 = \{a\}$ with $a, b \in A$. 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 multisets. We denote by τ the special empty label that is the empty multiset. 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 (number of nodes minus one) in the longest branch of the tree. Two trees $T_1 = (\mathcal{N}_1, E_1, \mathcal{A}_1)$ and $T_2 = (\mathcal{N}_2, E_2, \mathcal{A}_2)$ are *equal renaming of the nodes*; i.e. $T_1 = T_2$ iff $\mathcal{A}_1 = \mathcal{A}_2$ (the labels are the same), $|E_1| = |E_2|$, and there is a bijective function $rn : \mathcal{N}_1 \to \mathcal{N}_2$ such that $\forall (n, \alpha, m) \in E_1$ then $(rn(n), \alpha, rn(m)) \in E_2$.

Examples of rooted trees with labeled edges are given in Fig. 1:

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

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

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

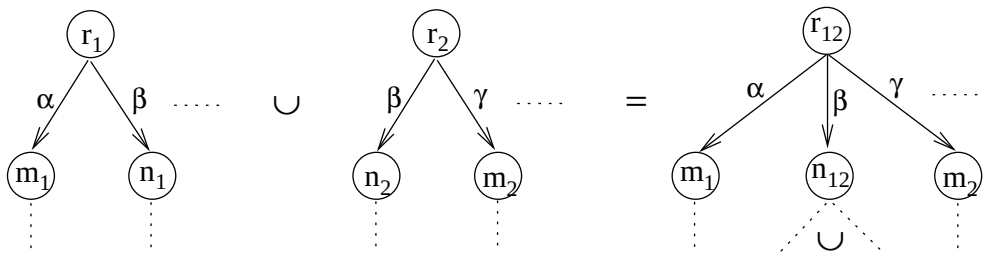

Figure 2: Join of two rooted trees.

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.

In the following we define some 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 Fig.2. 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. 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 Fig.3 illustrates this operation. To concatenate T_1 with T_2 follow the steps:

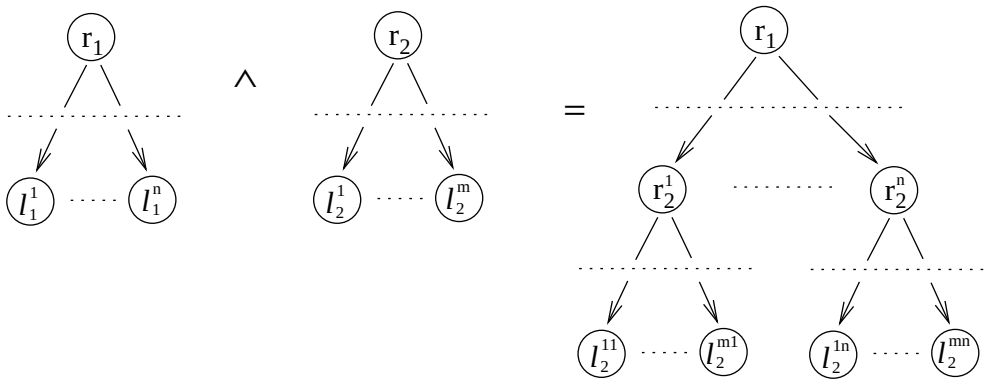

Figure 3: Concatenation of two rooted trees.

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 multisets). The procedure of concurrently joining two trees T_1 and T_2 taken as before consists in the following steps:

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.[4] Each new label $\alpha' = \alpha_1 \cup \alpha_2$ is the multiset union of the two component labels.

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

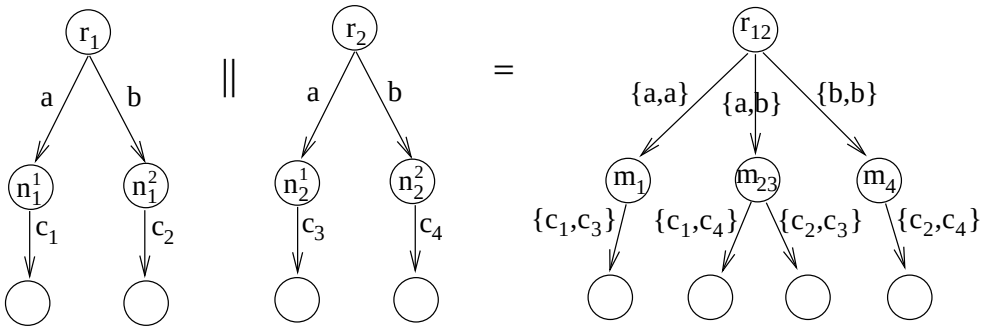

Figure 4: Example of concurrent join of two rooted trees.

(b) add a new edge (r_{12}, α', m) to E_{12} and the new node m to $\mathcal{N}_{12}$;

(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 Fig.4.

In the $\|$ operation we use the union of two labels which is the union of the two associated multisets. Note that the empty label τ "vanishes" when is joined with another label because τ is the empty multiset 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 multisets of basic labels) we just write $\{\alpha, \beta\}$. Moreover, the τ label is often omitted so we consider $\{\tau, \alpha, \beta\} = \{\alpha, \beta\}$.

For our purpose of giving a standard interpretation for $\mathcal{CA}$ in Section 3.2 we need to be able to interpret the special actions $\mathbf{1}$ and $\mathbf{0}$, 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 in Section 3.2.

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 A or τ. Thus the number of basic trees is $|\mathcal{RT}_B| = |A| + 2$.

We consider only *pruned trees*.

Definition 3.2 (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 branches that contain an edge labeled with τ that does not contain a black node remove the edge (in the usual way) unless 1c;*
   (c) *if (r, τ, n) with r the root node and $n : W$ is the only edge in the tree then do nothing.*
2. *for each black node n:*
   (a) *first remove the subtree with root n;*
   (b) *afterwords 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, 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 is called *pruning a tree* and 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 3.1. *Any pruned tree either contains no black nodes, or it is the tree with one root node r, one edge labeled with τ and ending in a black node $(r, \tau, n : B)$.*

Proof: The proof follows the steps in Definition 3.2 which deal with black nodes; i.e. steps 2, 3, and 4. The proof shows that these steps are suficient to eliminate all the black nodes in the trees. Step 2 is applied once in two stages by checking each node in the tree: in the first stage (corresponding to step 2a) all the branches of the tree which contain a black node are trimmed such that the black node is the last node in the branch. The second stage (corresponding to step 2b) takes care that there is no transition which ends in a black node and is labeled with a compound action (i.e. every edge ending in a black node is labeled with τ).

Step 3 is applied several times until the stopping condition is satisfied. The repeted application of step 3 is assured by step 4. The application of step 3 basically tries exhaustively to remove black nodes. The efective removing of the black nodes is done in step 3b. Step 3a shortens any branch containing a black node such that to reach a condition when to apply step 3b. The stopping condition is:

- Either there is no black node remained and thus the first statement of the proposition is satisfied.
- or there exists a black node n of an edge (m, τ, n) and there is no other edge (m, α, n') (i.e. no adjacent edge) and also there is no edge (r, α, m) (i.e. no edge before); and thus the second statement of the proposition is now satisfied.

□

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

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 3.1. 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 basic 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 induction, 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. $I_{\mathcal{CA}}(\alpha \& \beta) = I_{\mathcal{CA}}(\alpha) || I_{\mathcal{CA}}(\beta)$.

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 $\hat{I}_{\mathcal{CA}}(\alpha + \beta) = \hat{I}_{\mathcal{CA}}(\alpha) \cup \hat{I}_{\mathcal{CA}}(\beta)$.[5] This means that the function $Prune$ is not homomorphic which means that after composing two pruned rooted trees the function $Prune$ has to be applied again. On the other hand Lemmas 3.2, 3.3, and 3.7 give other useful properties of the $Prune$ function.

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

Lemma 3.2. *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 $Prune(T_1') = Prune(T_2')$.*

Proof: We do not have a complete formal proof of this result but we strongly believe in its intuitive validity and thus we conjecture it here. □

Lemma 3.3. *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
$Prune(T_1[op]T_2) = Prune(T_1'[op]T_2')$.

Proof: The proof considers three cases, one for each operator over the rooted trees. In each case it will analyze the behavior of each step in the *Prune* function of Definition 3.2.

Case 1 (for $\cup$). We need to prove that $Prune(T_1 \cup T_2) = Prune(T_1' \cup T_2')$. We need first to take a careful look at the operator $\cup$. It is clear that the operator does not change the labels of the edges of the old trees. Moreover, the $\cup$ operator just takes the sets of edges on each level of the two trees and puts them together acting only in the case when two edges are the same (are labelled the same).

We then investigate the *Prune* function to note that most of the steps are independent of the adjacent edges on each layer (only step 3 needs further investigation). For step 1a the *Prune* function takes each label of each edge and removes the τ and this operation may be done on parts of the tree, thus applying step 1a of *Prune* first to T_1 and then to T_2 is the same as applying step 1a of *Prune* directly to $T_1 \cup T_2$. The same discussion holds for step 1b.

We now use the proof principle *reductio ad absurdum* to finish the rest of the proof for this case. Therefore we try to prove the negate of the conclusion and get a contradiction. We consider that $Prune(T_1 \cup T_2) \neq Prune(T_1' \cup T_2')$ and thus $\exists(n, \alpha, m)$ with $m : W$ an edge which makes one pruned tree different from the other. Without loss of generality consider $(n, \alpha, m) \in Prune(T_1 \cup T_2)$. The discussion above about step 1 of *Prune* and the behavior of $\cup$ suggests that the edge should have come from one of the initial (unpruned) trees and thus either $(n, \alpha \cup \tau^*, m) \in T_1$ or $(n, \alpha \cup \tau^*, m) \in T_2$ where by $\alpha \cup \tau^*$ we mean that the initial label could have had, or not, a τ inside (which was removed by the *Prune*). The special case when $m : B$ is a black node is treated later.

Without loss of generality consider $(n, \alpha \cup \tau^*, m) \in T_1$. Thus, we apply $Prune(T_1) = \hat{T}_1$ and $Prune(T_1') = \hat{T}_1$, which means that $(n, \alpha, m) \in \hat{T}_1$, and

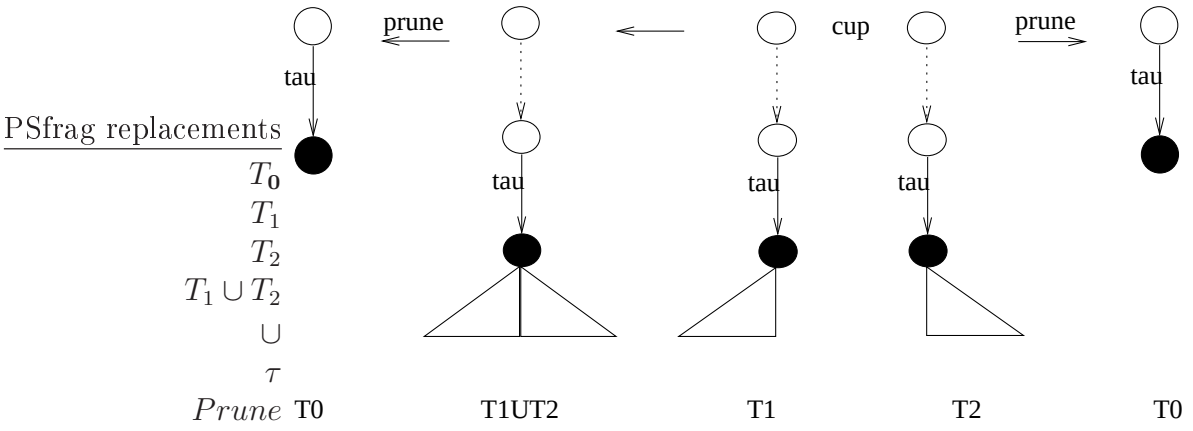

Figure 5: Example for the special case of the tree $T_1 \cup T_2$ which is pruned into $T_{\mathbf{0}}$.

thus we conclude that $(n, \alpha \cup \tau^*, m) \in T_1'$. Moreover, the edge $(n, \alpha \cup \tau^*, m)$ will still be in the join $T_1' \cup T_2'$ of the two trees. Because $m : W$ is a white node by $Prune(T_1' \cup T_2')$ the edge (n, α, m) will still be in the resulting pruned tree. This is a contradiction with the initial (wrong) assumption and thus the original conclusion of the case is true.

By Proposition 3.1 we know that by applying $Prune$ all the black nodes are removed with the exception when the remaining tree is the one representing $\mathbf{0}$. The special case mentioned before is when after pruning (without loos of generality) $Prune(T_1 \cup T_2) = T_{\mathbf{0}}$ where $T_{\mathbf{0}} = (\{r, n\}, \{(r, \tau, n)\}, \{\tau\})$ and $n : B$ is a black node. When analyzing the $Prune$ function we note that the only way to obtain such a pruned tree is if the original $T_1 \cup T_2$ is a tree formed of two trees concatenated such that the first tree is just a single path ending in a black node as in Figure 5 (i.e. the tree $T_1 \cup T_2$ starts with a single path which ends in a black node; and the black node is the root of the second tree). Therefore it means that both trees T_1 and T_2 have the same structure (i.e. starting with the single branch which ends in a black node). This means that both $Prune(T_1)$ and $Prune(T_1)$ are the tree $T_{\mathbf{0}}$. By the hypothesis we have that also $Prune(T_1')$ and $Prune(T_1')$ and thus $Prune(T_1' \cup T_2')$ are the tree $T_{\mathbf{0}}$. This is a contradiction to the initial (wrong) assumption and thus the original conclusion of the case is true.

Case 2 (for $\widehat{\ }$). We need to prove that $Prune(T_1 \widehat{\ } T_2) = Prune(T_1' \widehat{\ } T_2')$. We first look at the behavior of $\widehat{\ }$ which just appends the second tree to each leaf of the first tree. We again analyze the behavior of $Prune$ w.r.t. $\widehat{\ }$ operator.

Note that applying $Prune$ to $T_1 \widehat{} T_2$ behaves as applying first $Prune$ to the second tree T_2 and then applying $Prune$ to the first tree so that to obtain in both cases (i.e. $Prune(T_1 \widehat{} T_2)$ and $Prune(T_1' \widehat{} T_2')$) the same pruned tree. We still need to be careful to one thing. For each black node (cf. step 2) we have to remove the subtree. In this case we take a look at the first trees T_1 and T_1' which may have at the leafs different black nodes. Anyway, because we have the same pruned tree $\hat{T}_1$ which by Proposition 3.1 has no black nodes then we do not care which subtrees (corresponding to the different black nodes in the different trees T_1 and T_1') from the big concatenated trees are removed, as in the end we ramain with the same leafs for the upper part of the trees. The only remaining problem is when the pruned upper trees are the basic tree with one edge labeled by τ and ending in a black node. In this case the pruning of the whole big trees will end up in the same basic tree with one edge labeled by τ and ending in a black node (which models the action $\mathbf{0}$).

Case 3 (for $\|$). We need to prove that $Prune(T_1 \parallel T_2) = Prune(T_1' \parallel T_2')$ in the hypothesis that $Prune(T_1) = Prune(T_1')$ and $Prune(T_2) = Prune(T_2')$. Note that $\|$ operator manipulates the labels of the trees adding to the new tree new compound labels to the edges. This makes the investigation of this case more dificult. Note more, that the $\|$ operator is applied in stages on each level of the trees. Therefore we consider first the behavior of $Prune$ on each level w.r.t. $\|$ operator.

A fact is that after pruning of the small trees (T_1, T_1', T_2, T_2') their respective first levels will contain exactly the same edges with the same labels (i.e. T_1 with T_1', and T_2 with T_2'). A first question is: Knowing the fact above, is it the case that after pruning the big trees (i.e. $T_1 \parallel T_2$ and $T_1' \parallel T_2'$) we obtain at each level inductively the same edges labeled with the same compound labels?

We prove the question by *reductio ad absurdum*: suppose in one of the big trees there is one edge different (i.e. which does not appear in the other tree at the same level). Take edge $(r, \alpha, n) \in T_1 \parallel T_2$. Note also that after pruning there is no compound label which contains the τ. This fact is easily proven by looking at Definition 3.2. Therefore, α does not contain τ and is the result of union of two compound labels α_{T_1} and α_{T_2} of the same level in the two trees. The labels α_{T_1} and α_{T_2} are also sets of basic actions and do not contain τ. Note that $Prune$ does not add new basic actions to the labels of the trees, nor does it remove basic actions from the compound labels[6]. Therefore, $\alpha_{T_1} \cup \tau^*$ must have been a label of T_1 which because $Prune(T_1) = Prune(T_1')$ it means that α_{T_1} must be also a label of T_1' at the

[6] The removing is done only in the special steps and it removes labels all together with the edge.

PSfrag replacements

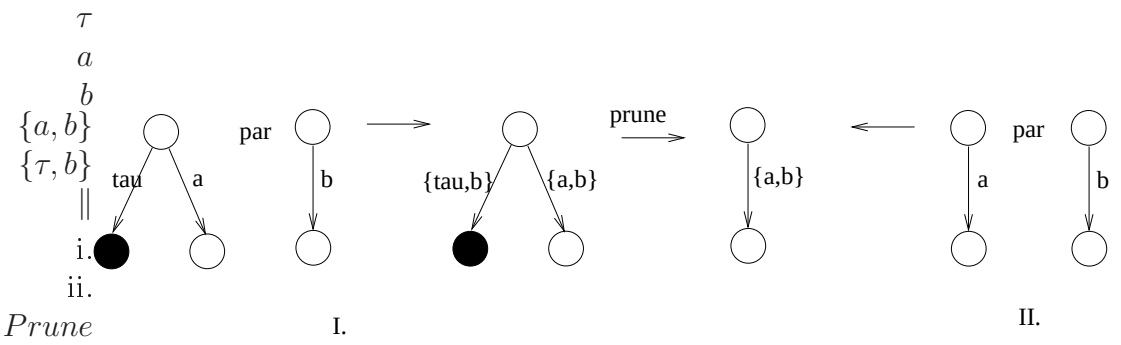

Figure 6: Example for Lemma 3.3.

same level. The same argument is valid for α_{T_2} which must be a label of T_2' at the same level. Therefore, in the compound tree $T_1' \parallel T_2'$ there must be at the same level a label formed just of α_{T_1}, α_{T_2} and possibly τ (which after pruning is removed). This is in contradiction with our supposition that there is no label α at the same level in the pruned tree of $T_1' \parallel T_2'$.

We want to analyze one more delicate case when at one level there is an edge labeled just with τ and it ends in a black node (the most simple case is pictured in Figure 6). One may say that this edge would insert in the big trees new and different edges labeled with $\tau \cup \alpha_{T_i}$ where α_{T_i} is a label of an edge in the other tree. This is true, but such an edge ends in a black node and is subject to the sequence of steps 2a, 2b, and 3b from the *Prune* function which removes the edge in the pruned tree.

This concludes the proof of the third case and of the lemma. □

Lemma 3.3 suggests the following result.

Corollary 3.4. *$\forall \alpha, \alpha', \beta, \beta' \in \mathcal{CA}$ 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 3.5 (Completeness of $\mathcal{CA}$ over $\mathcal{RT}$).
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 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:
$\mathcal{CA} \vdash \alpha = \beta \Rightarrow \hat{I}_{\mathcal{CA}}(\alpha) = \hat{I}_{\mathcal{CA}}(\beta)$ modulo renaming of the nodes.
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.

To make it more precise we define a relation $\doteq$ to denote the *equality modulo renaming of the nodes* between two rooted trees. Besides modulo renaming of the nodes the equality $\doteq$ is based on the usual equality on rooted trees where for example the branches of a tree are not ordered.

We consider the usual *basic rules* of equational reasoning which are *reflexivity*, *symmetry*, *transitivity*, and *substitution*.

We have as *basis* step of the induction the axioms of Table 1 and we take a case for each axiom.

For the next four cases related to axioms (1)-(4) we are looking at the $\cup$ operator on rooted trees. The main behavior of $\cup$ is that for each level of the tree it combines any two edges with the same label and proceeds the same for the subsequent levels. So, if we may regard the edges of one level of a tree as a set of edges one can easily see that $\cup$ makes the union of the edges.[7] We know that *union* for sets is associative, commutative and idempotent.

Case 1 (axiom (1)). Let $\alpha = \alpha_1 + (\alpha_2 + \alpha_3)$ and $\beta = (\alpha_1 + \alpha_2) + \alpha_3$. Because $I_{\mathcal{CA}}$ is homomorphic then $I_{\mathcal{CA}}(\alpha) = I_{\mathcal{CA}}(\alpha_1) \cup (I_{\mathcal{CA}}(\alpha_2) \cup I_{\mathcal{CA}}(\alpha_3))$ and $I_{\mathcal{CA}}(\beta) = (I_{\mathcal{CA}}(\alpha_1) \cup I_{\mathcal{CA}}(\alpha_2)) \cup I_{\mathcal{CA}}(\alpha_3)$. Following the discussion above, for $I_{\mathcal{CA}}(\alpha)$ the $\cup$ operator, for each level of the trees first identifies the common edges of $I_{\mathcal{CA}}(\alpha_2)$ and $I_{\mathcal{CA}}(\alpha_3)$ and combines them and afterwords combines the remaining edges with the common ones of $I_{\mathcal{CA}}(\alpha_1)$ and proceeds to the next levels. This behavior results in the same edges for the $I_{\mathcal{CA}}(\beta)$ where first the common edges of $I_{\mathcal{CA}}(\alpha_1)$ and $I_{\mathcal{CA}}(\alpha_2)$ are identified, which are part of the edges identified in the second combination for the $I_{\mathcal{CA}}(\alpha)$ before. In the next step $\cup$ combines the edges also common to $I_{\mathcal{CA}}(\alpha_3)$.

Case 2 (axiom (2)). For commutativity it is simple as $\cup$ has also a commutative behavior and also think of the discussion before.

Case 3 (axiom (3)). Form commutativity it is simple to see the first equality $\alpha + \mathbf{0} = \mathbf{0} + \alpha$. The second equality $\alpha + \mathbf{0} = \alpha$ is treated at the level of *Prune* function. Note that the tree $I_{\mathcal{CA}}(\mathbf{0})$ is the tree with one edge labeled with τ and ending in a *black* node. For this the combination $I_{\mathcal{CA}}(\alpha) \cup I_{\mathcal{CA}}(\mathbf{0})$ gives the tree in Figure 7ii. which when applying the *Prune* function it is applied the step 3b which removes the newly added edge corresponding to $I_{\mathcal{CA}}(\mathbf{0})$ thus resulting in the same pruned tree and so $\hat{I}_{\mathcal{CA}}(\alpha + \mathbf{0}) = \hat{I}_{\mathcal{CA}}(\alpha)$.

Case 4 (axiom (4)). For the idempotence it is simple to see that when joining the tree $I_{\mathcal{CA}}(\alpha_1)$ with itself we obtain the same tree as all the edges of one of the trees are removed as being equal with the edges of the first tree.

[7] Where remember that for sets the union keeps only one copy of each element.

PSfrag replacements

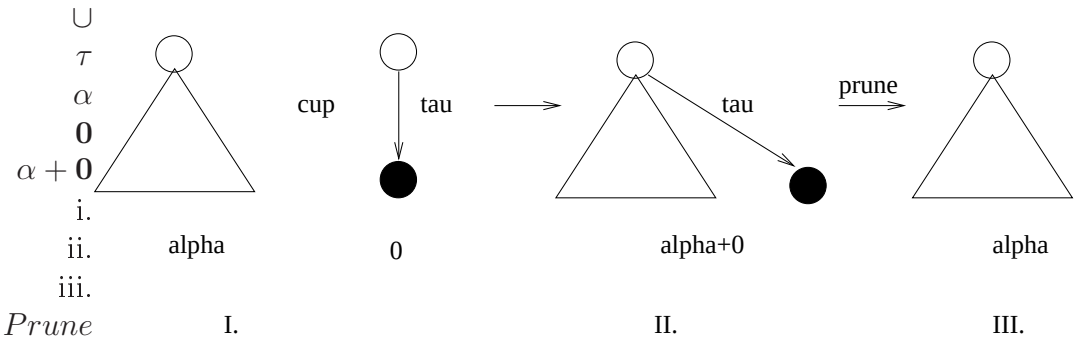

Figure 7: Example for Case 3.

Case 5 (axiom (5)). This case for $\alpha \cdot (\beta \cdot \gamma) = (\alpha \cdot \beta) \cdot \gamma$ is simple and we leave it to the reader. Basically it does not matter in which order the trees are joined one to the end of the other.

Case 6 (axiom (6)). This case is also treated at the level of the *Prune* function by the step 1b of the Definition 3.2 which removes the edges labeled with τ and ending in a white node. This would correspond to the tree $I_{\mathcal{CA}}(\mathbf{1})$. Thus we have $\hat{I}_{\mathcal{CA}}(\alpha \cdot \mathbf{1}) = \hat{I}_{\mathcal{CA}}(\alpha)$.

Case 7 (axiom (7)). This case is proven at the level of *Prune* function by considering steps 2–3a. The rest of the proof is pictured in Figure 8.

For the part $\mathbf{0} \cdot \alpha = \mathbf{0}$ we have to concatenate trees $I_{\mathcal{CA}}(\mathbf{0})$ and $I_{\mathcal{CA}}(\alpha)$ into the tree in Figure 8i. The function *Prune* applies step 2a which removes the subtree starting in a black node and we obtain the tree $T_{\mathbf{0}}$ and thus $\hat{I}_{\mathcal{CA}}(\mathbf{0} \cdot \alpha) = \hat{I}_{\mathcal{CA}}(\mathbf{0})$.

For the part $\alpha \cdot \mathbf{0} = \mathbf{0}$ we concatenate to the tree $I_{\mathcal{CA}}(\alpha)$ the tree $I_{\mathcal{CA}}(\mathbf{0})$, which means that at each leaf node of tree $I_{\mathcal{CA}}(\alpha)$ we attach the tree $I_{\mathcal{CA}}(\mathbf{0})$ which is formed of only one edge labelled with τ and ends in a black node. A simple example for this second part of the case is pictured in Figure 8ii. It is simple to see that function *Prune* applies step 3a for the black nodes at the leafs of the tree and removes the immediate upper edges (from the tree $I_{\mathcal{CA}}(\alpha)$). In the next round step 3b is applied. This process of applying the sequence of steps 3a and 3b goes up the tree until it removes all the edges and remains with the tree $I_{\mathcal{CA}}(\mathbf{0})$.

Case 8 (axiom (8)). Take $\alpha = \alpha_1 \cdot (\alpha_2 + \alpha_3)$ and $\beta = \alpha_1 \cdot \alpha_2 + \alpha_1 \cdot \alpha_3$. Because $I_{\mathcal{CA}}$ is homomorphic then $I_{\mathcal{CA}}(\alpha) = I_{\mathcal{CA}}(\alpha_1) \widehat{} I_{\mathcal{CA}}(\alpha_2 + \alpha_3)$ and $I_{\mathcal{CA}}(\beta) = I_{\mathcal{CA}}(\alpha_1 \cdot \alpha_2) \cup I_{\mathcal{CA}}(\alpha_1 \cdot \alpha_3)$. Therefore T_α is composed first of the tree T_{α_1} and at each leaf it is concatenated the tree $T_{\alpha_2+\alpha_3}$. On the other

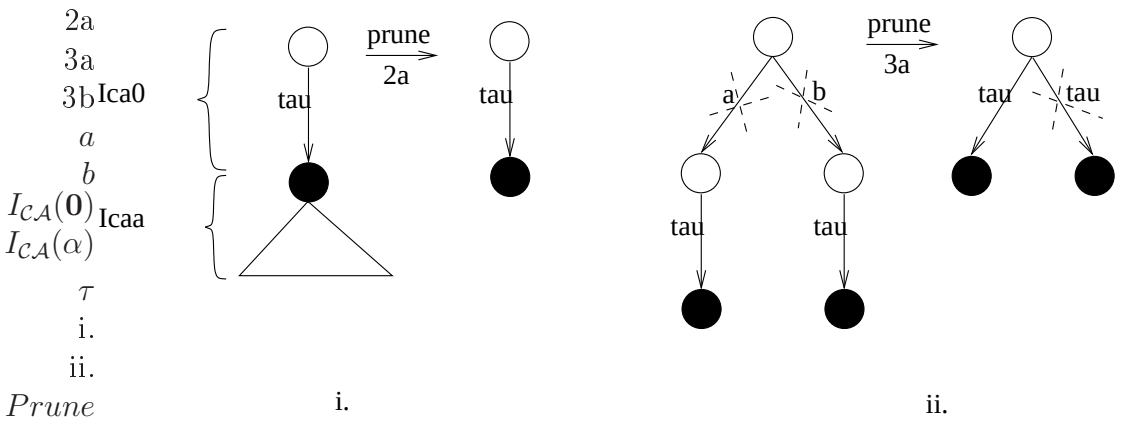

Figure 8: Illustration of Case 7.

hand tree T_β has to be a joint of trees $T_{\alpha_1 \cdot \alpha_2}$ and $T_{\alpha_1 \cdot \alpha_3}$, which when joining the suffix trees T_{α_1} which are equal we obtain the same tree T_{α_1} at the top followed at the leafs by the joining of the remaining trees T_{α_2} and T_{α_3}. Thus giving the same tree. Pruning the same tree gives the same pruned tree and thus $\hat{I}_{\mathcal{CA}}(\alpha) = \hat{I}_{\mathcal{CA}}(\beta)$.

The case for axiom (9) is treated similarly.

Case 9 (axiom (10)). We need to prove that $\hat{I}_{\mathcal{CA}}(\alpha \& (\beta \& \gamma)) = \hat{I}_{\mathcal{CA}}((\alpha \& \beta) \& \gamma)$. For this case it is suficient to consider only $I_{\mathcal{CA}}$ and thus we need to prove that $I_{\mathcal{CA}}(\alpha) \parallel (I_{\mathcal{CA}}(\beta) \parallel I_{\mathcal{CA}}(\gamma)) = (I_{\mathcal{CA}}(\alpha) \parallel I_{\mathcal{CA}}(\beta)) \parallel I_{\mathcal{CA}}(\gamma)$ (as $I_{\mathcal{CA}}$ is homomorphic). The behavior of the $\parallel$ operator is to make the cartesian product of the labels of the trees on each level and then descends recursively to the other levels to apply the same concurrent join operation. Because the cartesian product is associative we obtain on each level of the trees the same edges and for each new edge we use inductively the hypothesis to obtain the same subtrees. Thus we have the desired conclusion.

Case 10 (axiom (11)). Follows immediately from the commutativity of $\parallel$ operator; i.e. the fact that the order of the edges of the trees does not matter.

Case 11 (axiom (12)). This case is proven at the level of *Prune* function by the step 1a which removes the τ from the multiset labels thus resulting in the same pruned tree.

Case 12 (axiom (13)). This case is proven at the level of *Prune* function also by considering steps 2-3a. This is because **0** introduces *black* notes into

the tree $I_{\mathcal{CA}}(\alpha)$.

Case 13 (axiom (14)). For this proof we do not need to take into consideration the *Prune* function application as we can manage to show the following: $I_{\mathcal{CA}}(\alpha) \parallel (I_{\mathcal{CA}}(\beta) \cup I_{\mathcal{CA}}(\gamma)) = (I_{\mathcal{CA}}(\alpha) \parallel I_{\mathcal{CA}}(\beta)) \cup (I_{\mathcal{CA}}(\alpha) \parallel I_{\mathcal{CA}}(\gamma))$. It is known that for sets the cartesian product is distributive over the union of sets. Therefore, in our case we get in both big trees the same new edges at each level. We need to take care only of the common parts of the trees. In the left tree the $\cup$ operator joins the two subtrees for each common edges and afterword the edge is combined with the edges in the tree $I_{\mathcal{CA}}(\alpha)$ and the subtrees are also combined thus offering the oportunity to apply inductively the same reasoning. For the tree on the right first the $\parallel$ operator combines the edges of the trees and because of the fact above we will have the same set of identical edges which will join the subtrees (obtained by concurrent composition $\parallel$). Thus, applying the hypothesis inductively we obtain the same trees.

The case for axiom (15) is treated similarly.

Case 14 (axiom (16)). The proof is natural as it basically states that the concurrent join $\parallel$ operates on levels of the tree and then descends to the subsequent levels until no more join is possible and from that point on the remaining part of the tree is just copied.

For the inductive step we have the following cases corresponding to the derivation rules:

Case 1 (reflexivity). $\mathcal{CA} \vdash \alpha = \alpha$ then because $\hat{I}_{\mathcal{CA}}$ is a function it is imeediate by the definition that $\hat{I}_{\mathcal{CA}}(\alpha) = \hat{I}_{\mathcal{CA}}(\alpha)$ and thus $\hat{I}_{\mathcal{CA}}(\alpha) \doteq \hat{I}_{\mathcal{CA}}(\alpha)$.

Case 2 (symmetry). If $\mathcal{CA} \vdash \alpha = \beta$ then $\mathcal{CA} \vdash \beta = \alpha$, where by the induction hypothesis we have that from $\mathcal{CA} \vdash \alpha = \beta$ implies that $\hat{I}_{\mathcal{CA}}(\alpha) = \hat{I}_{\mathcal{CA}}(\beta)$. We know that the equality $\doteq$ on rooted trees is symmetric and thus we have the conclusion $\mathcal{CA} \vdash \beta = \alpha \Rightarrow \hat{I}_{\mathcal{CA}}(\beta) \doteq \hat{I}_{\mathcal{CA}}(\alpha)$.

Case 3 (transitivity). If both $\mathcal{CA} \vdash \alpha = \beta$ and $\mathcal{CA} \vdash \beta = \gamma$ then $\mathcal{CA} \vdash \alpha = \gamma$. The proof argument is similar to that in Case 2 and is based on the fact that $\doteq$ is transitive.

Case 4 (substitution). We must consider each of the three operators on action $+, \cdot, \&$. We look only at $+$ where if $\mathcal{CA} \vdash \alpha = \alpha'$ and $\mathcal{CA} \vdash \beta = \beta'$ then $\mathcal{CA} \vdash \alpha + \alpha' = \beta + \beta'$. By the induction hypothesis we have that $\hat{I}_{\mathcal{CA}}(\alpha) = \hat{I}_{\mathcal{CA}}(\alpha')$ and $\hat{I}_{\mathcal{CA}}(\beta) = \hat{I}_{\mathcal{CA}}(\beta')$ which by Lemma 3.3 we have our

conclusion $\hat{I}_{\mathcal{CA}}(\alpha + \alpha') \doteq \hat{I}_{\mathcal{CA}}(\beta + \beta')$.

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. 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 the inverse function another action α'. The obtained action α' has to be equal by the axiom system with the original action $\alpha = \alpha'$. Having this isomorphism then from two actions α and β we get the same tree T_γ from where we translate back to the same action $\gamma = \alpha = \beta$ which is our conclusion.

Remember that the term algebra $T_{\mathcal{CA}}$ is free in the class of algebras over the generators $\mathcal{A}_B$. The fact that $\hat{I}_{\mathcal{CA}}$ is an algebraic isomorphism makes the $\mathcal{RT}$ algebra also free in the class of algebras $\mathcal{CA}$, which means that any property on the rooted trees holds on the action terms and the converse.

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

Definition 3.3. *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 based on the deduction rules and we leave it to the reader.

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

Lemma 3.6 (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 $\exists \alpha \in \mathcal{CA}$
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 $I_{\mathcal{CA}}^{-1}$ 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 $\hat{I}_{\mathcal{CA}}^{-1}$ 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 $I_{\mathcal{CA}}^{-1}$ to the tree operators is natural:

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

With this construction we have proven that $I_{\mathcal{CA}}^{-1}$ is defined on the whole domain $\mathcal{RT}$ and thus $\hat{I}_{\mathcal{CA}}^{-1}$ 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 $\hat{I}_{\mathcal{CA}}^{-1}$ 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 ($\hat{T}_1 = \hat{T}_2$) than the function $\hat{I}_{\mathcal{CA}}^{-1}$ 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 $\hat{I}_{\mathcal{CA}}^{-1}$ is a well defined function.

Lemma 3.7. *Function Prune preserves the relation $\equiv$ on actions, meaning that $\forall T \in \mathcal{RT}$ if $Prune(T) = \hat{T}$ then $I_{\mathcal{CA}}^{-1}(T) \equiv I_{\mathcal{CA}}^{-1}(\hat{T})$.*

Proof: The proof uses induction on the pruned tree and thus considers a case for each step in the Definition 3.2 of the *Prune* function. We investigate how *Prune* changes the tree and how these changes take the action corresponding to the initial tree into a equivalent action w.r.t. $\equiv$ relatio.

The *basis* of the induction considers only basic trees $\mathcal{RT}_B$ for which the *Prune* function returns the same tree (i.e. has no efect on the initial tree) and thus $I^{-1}_{\mathcal{CA}}(T) = I^{-1}_{\mathcal{CA}}(Prune(T)) \Rightarrow I^{-1}_{\mathcal{CA}}(T) \equiv I^{-1}_{\mathcal{CA}}(Prune(T))$.

The *inductive step* considers compound trees and makes use of the homomorphic definition of $I^{-1}_{\mathcal{CA}}$ considering a case for each step of the Definition 3.2.

- **Case for step 1a** when the τ is removed from the compound labels. In this case we consider the initial tree changed only by step 1a which means that there are no black nodes (and thus steps 2-4 are not applied) and also there are no edges labeled just with τ (necessary for step 1b). In this case relevant is composition of trees by means of $\|$ operator. $I^{-1}_{\mathcal{CA}}(T) \equiv I^{-1}_{\mathcal{CA}}(\hat{T})$ because of the axiom (12) of Table 1. Thus, for a chain of applications of the step 1a the old action $I^{-1}_{\mathcal{CA}}(T)$ becomes an equivalent action $I^{-1}_{\mathcal{CA}}(\hat{T})$ because of a chain of application of axiom (12).

- **Case for step 1b** is related to axiom (6). Now we consider inductively also pruned trees obtained by applying step 1a. Note that for one application of step 1b from the old action it is obtained a new action which is equivalet because of the application of axiom (6). This is because the edge labeled just with τ and ending in a white node would have been translated by $I^{-1}_{\mathcal{CA}}$ into the $\mathbf{1}$ action. This small tree is concatenated with the big one which because of the homomorphism property of $I^{-1}_{\mathcal{CA}}$ it is related to the sequence composition $\alpha \cdot \mathbf{1}$ or $\mathbf{1} \cdot \alpha$ at the level of the actions. In conclusion, from a chain o applications of step 1b it is obtained by a chain of axiom (6) an equivalent action.

- **Case for step 1c** takes care that the tree which interprets $\mathbf{0}$ is a pruned tree (the edge is not removed).

  We now consider black nodes, and we look at one black node at a time and apply the sequence of steps from 2 onwords.

- **Case for step 2** is related to axiom (7) and (13). This step basically states that once entered into a black node it si the same as saying at the level of the actions that a fail $\mathbf{0}$ has occured, and thus axiom (7) is the case. First step 2a removes the tree below the black node (at the action level is equivalent to $\mathbf{0} \cdot \alpha = \mathbf{0}$) and then in step 2b it transforms the edge (by changing the label into τ) such that it is translated by $I^{-1}_{\mathcal{CA}}$ into $\mathbf{0}$ action.

- **Case for step 3a** is related to part $\alpha \cdot \mathbf{0}$ of axiom (7) as it moves up the branches of the tree the special edge $(n, \tau, m : B)$ which is transformed by $I_{\mathcal{CA}}^{-1}$ into $\mathbf{0}$.

  Note that the last two cases are also related to axiom (13) because the big tree may be result of concurrent join between $I_{\mathcal{CA}}(\alpha)$ and $I_{\mathcal{CA}}(\mathbf{0})$.

- **Case for step 3b** is related to axiom (3). Basically the *Prune* function removes the edges $(n, \tau, m : B)$ from the tree the same as axiom (3) removes $\mathbf{0}$ actions from choices.

  Note that the last two steps are applied repetedly, which at the level of the actions it is the same as moving the $\mathbf{0}$ action through the action.

□

From Lemma 3.7 we conclude the following useful congruences:

$$I_{\mathcal{CA}}^{-1} \circ I_{\mathcal{CA}} \equiv I_{\mathcal{CA}}^{-1} \circ \hat{I}_{\mathcal{CA}} \tag{21}$$

which by restriction implies

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

The proof of part 3 of Lemma 3.6 uses structural induction on the structure of the action α. Proving part 3 we prove that $\hat{I}_{\mathcal{CA}}^{-1}$ is the isomorphic image of $\hat{I}_{\mathcal{CA}}$ up to the congruence on actions $\equiv$.

Basis:

- For $\alpha = a$. $I_{\mathcal{CA}}(a) = (\{r, n\}, \{(r, a, n)\}, \{a\})$ with $n : W$ which by pruning remains the same tree and by the inverse $\hat{I}_{\mathcal{CA}}^{-1}$ we have the same action a.

- For $\alpha = \mathbf{1}$ or $\alpha = \mathbf{0}$ the same as above applies.

Inductive step:

- For $\alpha = \alpha_1 + \alpha_2$.

  By (22) we have that $\hat{I}_{\mathcal{CA}}^{-1} \circ \hat{I}_{\mathcal{CA}}(\alpha) \equiv I_{\mathcal{CA}}^{-1} \circ I_{\mathcal{CA}}(\alpha) = I_{\mathcal{CA}}^{-1}(I_{\mathcal{CA}}(\alpha_1 + \alpha_2))$ where because $I_{\mathcal{CA}}$ is a homomorphism we have that it is equal to $I_{\mathcal{CA}}^{-1}(I_{\mathcal{CA}}(\alpha_1) \cup I_{\mathcal{CA}}(\alpha_2))$ where by applying the homomorphic definition of $I_{\mathcal{CA}}^{-1}$ we get that $I_{\mathcal{CA}}^{-1} \circ I_{\mathcal{CA}}(\alpha) = I_{\mathcal{CA}}^{-1}(I_{\mathcal{CA}}(\alpha_1)) + I_{\mathcal{CA}}^{-1}(I_{\mathcal{CA}}(\alpha_2))$. Again by equation (22) we have that $I_{\mathcal{CA}}^{-1}(I_{\mathcal{CA}}(\alpha_1)) \equiv \hat{I}_{\mathcal{CA}}^{-1}(\hat{I}_{\mathcal{CA}}(\alpha_1))$ and $I_{\mathcal{CA}}^{-1}(I_{\mathcal{CA}}(\alpha_2)) \equiv \hat{I}_{\mathcal{CA}}^{-1}(\hat{I}_{\mathcal{CA}}(\alpha_2))$. By the inductive hypothesis we know

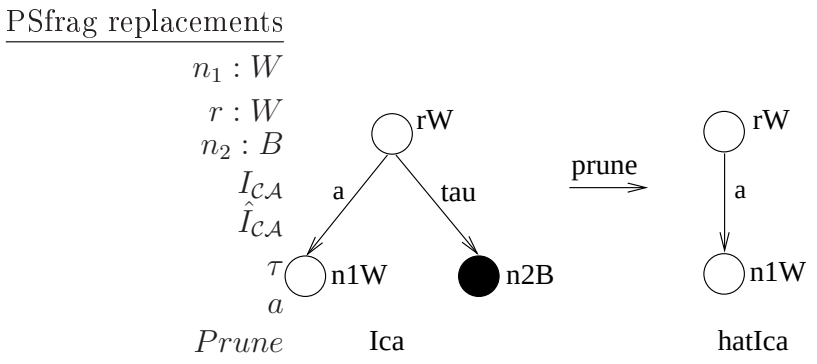

Figure 9: Example of applying the isomorphism $\hat{I}_{\mathcal{CA}}$.

that $\hat{I}^{-1}_{\mathcal{CA}} \circ \hat{I}_{\mathcal{CA}}(\alpha_1) = \alpha'_1 \equiv \alpha_1$ and that $\hat{I}^{-1}_{\mathcal{CA}} \circ \hat{I}_{\mathcal{CA}}(\alpha_2) = \alpha'_2 \equiv \alpha_2$. Which is equivalent to saying that $\mathcal{CA} \vdash \alpha'_1 = \alpha_1$ and $\mathcal{CA} \vdash \alpha'_2 = \alpha_2$ which by the substitution rule of equational reasoning we have that $\mathcal{CA} \vdash \alpha'_1 + \alpha'_2 = \alpha_1 + \alpha_2$ which is our desired conclusion; i.e. $\hat{I}^{-1}_{\mathcal{CA}} \circ \hat{I}_{\mathcal{CA}}(\alpha) \equiv^* \alpha'_1 + \alpha'_2 \equiv \alpha_1 + \alpha_2 = \alpha$.

- For $\alpha = \alpha_1 \cdot \alpha_2$ or $\alpha = \alpha_1 \& \alpha_2$ the reasoning is similar.

Consider as an example the special case when $\alpha = a + \mathbf{0}$ which is pictured in Figure 9. $I_{\mathcal{CA}}(\alpha) = (\{r, n_1, n_2\}, \{(r, a, n_1), (r, \tau, n_2)\}, \{a, \tau\})$ 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 1 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. $\hat{I}_{\mathcal{CA}}(\alpha) = \hat{I}_{\mathcal{CA}}(\beta) \Rightarrow \mathcal{CA} \vdash \alpha = \beta$ we make use of Lemma 3.6. From $\hat{I}_{\mathcal{CA}}(\alpha) = \hat{I}_{\mathcal{CA}}(\beta)$ 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 3.6. 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.

4 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 $\mathcal{A}_\mathbf{1}$; 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*[8]; 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 allways 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 allways succeeds so the action α can allways 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*

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

[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 guarded string is an alternation of tests and actions:

$$\phi_0?\ \alpha_1\ \phi_1? \ldots \alpha_n\ \phi_n? \tag{23}$$

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 (16) 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 4.1. *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 1 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 4.1 (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.*

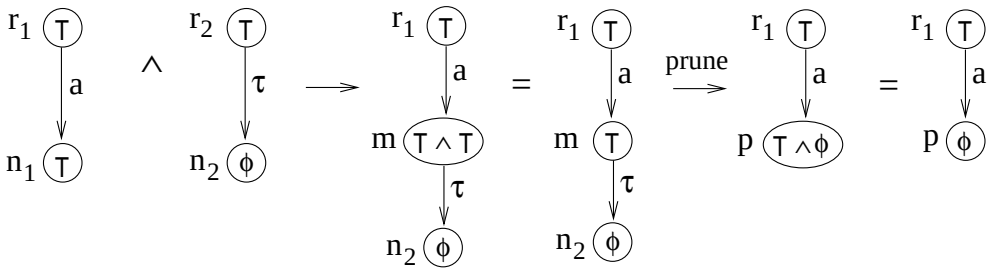

Figure 10: Example of concatenation of two guarded rooted trees.

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

In this section we extend the rooted trees of Section 3.1 with tests and call them *guarded rooted trees*. On this trees we give the standard interpretation of the $\mathcal{CAT}$ algebra.

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 4. 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 4.2. *We give in Fig. 10 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 afterwords 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 3.2 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 $\mathbf{1}$ and $\mathbf{0}$ 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 3.5 for $\mathcal{CA}$ algebra.

5 Canonic Form of Actions

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 canonical 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 5.1 (canonical form for $\mathcal{CAT}$). *For actions α defined with the operators $+$, $\cdot$, $\&$, and tests we have a canonical form denoted by ACF^{α} and defined as*

$$\alpha = \underset{\rho \in R}{+} \rho \cdot \alpha'$$

Where R contains elements either from basic actions, concurrent actions, or tests, and α' is a compound action in canonical form.

Theorem 5.1. *For every action α of the algebra $\mathcal{CAT}$ we have a corresponding ACF^{α}.*

Proof: We use structural induction on the structure of the actions of $\mathcal{A}$ given by the constructors of the algebra. In the inductive proof we take one case for each action construct. The proof also makes use of the equations of the algebra.

Basis:

a) If α is a base action a of $\mathcal{A}_B$ it is immediately proven to be in canonical form just by looking at the definition of the canonical form. Action a is a canonical form with the set R containing only one element, namely a and the $\cdot$ constructor is applied to a and to skip action $\mathbf{1}$ ($a \cdot \mathbf{1} = a$). Note that we appeal to the common sense and the choice ($+$) of only one action ($+_a\, a$) should be understood as choice among fail action $\mathbf{0}$ and a ($+_a\, a \stackrel{def}{=} \mathbf{0} + a = a$).

b) If $\alpha = \phi?$ is a test of $\mathcal{B}$ then it is considered in ACF^{α} by definition. This case is similar to the one for base actions. The set R of the canonical form contains only one test, and the action α' is the skip action which is in canonical form.

c) The special actions $\mathbf{1} = \top?$ and $\mathbf{0} = \bot?$ are given by tests and therefore are considered by definition to be in canonical form.

In the inductive step we consider only one step of the application of the constructors; the general compound actions should follow from the associativity of the constructors.

Inductive steps:

a) If $\alpha = \beta + \beta'$ is a compound action obtained by applying once the $+$ constructor. By the induction supposition β and β' are in canonical form. It means that β should be $\beta = +_{b_i} b_i \cdot \beta_i$ and $\beta' = +_{b'_j} b'_j \cdot \beta'_j$. Because of the associativity and commutativity of $+$, $\beta + \beta'$ is also in canonical form:

$$\beta + \beta' = \underset{b_i \in B}{+} b_i \cdot \beta_i \ + \ \underset{b'_j \in B'}{+} b'_j \cdot \beta'_j = \underset{a \in B \cup B'}{+} a \cdot \beta_a$$

where a and β_a are related in the sense that if $a = b_i$ then $\beta_a = \beta_i$ and if $a = b'_j$ then $\beta_a = \beta'_j$. Because the inductive hypothesis states that all β_i and β'_j are in canonical form it follows that also β_a (which is just a change of notation) is in canonical form.

b) If $\alpha = \beta \cdot \beta'$ with $\beta = +_{b_i} b_i \cdot \beta_i$ and $\beta' = +_{b'_j} b'_j \cdot \beta'_j$ in canonical form. We now make use of the distributivity of $\cdot$ over $+$, and of the associativity of $\cdot$ and $+$ constructors. α transforms in several steps into a canonical form. In the first step α is:

$$\alpha = \beta \cdot \beta' = (\underset{b_i \in B}{+} b_i \cdot \beta_i) \ \cdot \ (\underset{b'_j \in B'}{+} b'_j \cdot \beta'_j)$$

and if we consider $|B| = m$ then α becomes:

$$\alpha = b_1 \cdot \beta_1 \cdot (\underset{b'_j \in B'}{+} b'_j \cdot \beta'_j) \ + \ \ldots \ + \ b_m \cdot \beta_m \cdot (\underset{b'_j \in B'}{+} b'_j \cdot \beta'_j)$$

Subsequently α distributes the $\cdot$ over all the members of the choice actions. In the end α becomes a choice of sequences; when we consider $|B| = m$ and $|B'| = k$.

$$\alpha = b_1 \cdot \beta_1 \cdot b'_1 \cdot \beta'_1 \ + \ \ldots \ + \ b_m \cdot \beta_m \cdot b'_k \cdot \beta'_k$$

This is clearly a canonical form because all actions $\beta_i \cdot b'_j \cdot \beta'_j$ are in canonical form due to the inductive hypothesis.

c) If $\alpha = \beta \& \beta'$ with $\beta = +_{b_i} b_i \cdot \beta_i$ and $\beta' = +_{b'_j} b'_j \cdot \beta'_j$ in canonical form. The proof of this case is fairly lengthy and we show here only a simple particular case.

Let us consider actions $\beta = b \cdot \beta'$, $\gamma = c \cdot \gamma'$, and $\delta = d \cdot \delta'$ in canonical form. They are the components of $\alpha = (\beta + \gamma)\&\delta$. We apply the distributivity of & with respect to + and get:

$$\alpha = \beta\&\delta + \gamma\&\delta = (b \cdot \beta')\&(d \cdot \delta') + (c \cdot \gamma')\&(d \cdot \delta')$$

By applying equation (10) we get:

$$\alpha = b\&d \cdot \beta'\&\delta' + c\&d \cdot \gamma'\&\delta'$$

This shows that α is in canonical form because by the inductive supposition $\beta'\&\delta'$ and $\gamma'\&\delta'$ are in canonical form.

□

5.1 Action negation

One of the purposes of the investigation of the algebra in this paper 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.

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

$$\overline{\alpha} = \overline{\underset{\rho \in R}{+} \rho \cdot \alpha'} = \underset{b \in \overline{R}}{+} b \; + \; \underset{\rho \in R}{+} \rho \cdot \overline{\alpha'}$$

where ρ is either a basic action, a concurrent action, or a test. α' is a compound action in $ACF^{\alpha'}$. The set $\overline{R}$ is defined in the following.

Discussion: We elaborate here on a discussion we had [PS07b] about the nature of actio 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 allways do an action. It is simple to model passivity by action $\mathbf{1}$ *skip*. Moreover, in a subsequent paper we add time for actions (i.e. the duration of an action) and with time it is natural to model idleing 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...

We include action negation as a restricted operator of the $\mathcal{CAT}$ algebra. Action negation is restricted to being applied only once in an action, i.e. we cannot find negation applied to the negation of an action (e.g. $\overline{\overline{a}}$ or $\overline{a+\overline{b}}$). On the other hand, we can have combination using the normal operators of negated actions and normal actions (e.g. $a+\overline{b}$).

In the construction of the set $\overline{R}$ we include several things. First we look at the negation of a single test (i.e. when ρ is the test $\phi?$) $\overline{\phi?}$ which is just the negated test in the Boolean algebra $(\neg\phi)?$. The problematic part is the negation of a basic action a. For example, another basic action b different from a is part of $\overline{a}$, but also any concurrent composition with itself $(b\&b, \ldots b\&b\&b \ldots)$ is part of $\overline{a}$. This gives an infinite number of actions because operator $\&$ is not idempotent.

We considered any action of $\mathcal{CAT}$ to be finite as it is constructed by applying the constructors of the algebra a finite number of times. For this we also considered finite rooted trees. Note that because of the action negation and of the non-idempotence of $\&$ we get infinite actions and rooted trees with infinite branching. We try to avoid this problem by giving a procedure for generating the actions of $\overline{R}$ using the demanding order $<_{\&}$ of the algebra, and by defining *tree schemas* for interpreting the action negation.

Let us take one more 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 $c, c\&c, c\&d \in \overline{a+b}$. Formally, any concurrent action $c \in \mathcal{A}_{\&}$ with the properties that $a \not<_{\&} c$ and $b \not<_{\&} c$ is a negation of $a+b$.

We define the set $\overline{R}$ to contain:

1. $\{(\neg\phi)? \,|\, \phi \in R\}$; all the negation of the tests in R.
2. $\{\alpha \,|\, \alpha \in \mathcal{A}_{\&}, \text{ and } \forall\beta \in R, \beta \not<_{\&} \alpha\}$; all actions α (constructed using

only &) with the property that there is no action β of R which is less than α with respect to the demanding order $<_{\&}$.

This definition still generates an infinite set $\overline{R}$ which means that we still have infinite branching in the rooted tree associated to the action negation. The infiniteness of the action negation is not so problematic as it can be characterized. We have infinite branching because whenever we can put in the set $\overline{R}$ a compound action, e.g. $a\&b$ we have to put also all the actions $a\&b\&b\&\ldots$ and more. With this observation we can characterize the infinite set $\overline{R}$ as a finite set in terms of *action schemas*. An action schema is defined with respect to basic actions and is denoted $a|_k^\infty$. The action schema represents the choice between an infinite number of concurrent actions: $\forall a \in \mathcal{A}_B$ the action schema $a|_1^\infty = a + a\&a + a\&a\&a + \ldots$; the general definition $a|_k^\infty$ starts with the action $\underbrace{a\&\ldots\&a}_{k}$ and continues with larger actions. An action schema $a|_1^\infty$ may be executed concurrently with another action b and results in a new schema $b\&a|_1^\infty = b\&a + b\&a\&a + \ldots$.

Tree schemas are introduced to interpret the action schemas. A tree schema is a guarded rooted tree with special *edge schemas*. An edge schema is a special edge labeled with an action schema. An edge schema $(n, a|_1^\infty, m)$ represents an infinite set of normal edges $\{(n, \alpha, m) \,|\, \alpha \in a|_1^\infty\}$ where α is one of the actions in the infinite choice action represented by $a|_1^\infty$. Note that a tree schema is a finite representation of a guarder rooted tree with infinite branching.

Note that as expected the negation of an action is also in ACF^α. We conjecture that if we equip Propositional Dynamic Logic with such an action negation we still have decidability.

6 Relation between $\mathcal{CAT}$ and $\mathcal{CL}$

In this section we give a direct semantics to the $\mathcal{CL}$ language using the $\mathcal{CAT}$ algebra.

We start by defining some preliminary notions. Consider standard *labelled Kripke structures* and guarded rooted trees. We give now the definition of labeled Kripke structure that we use.[9]

Definition 6.1 (Labelled Kripke Structure). *A labeled Kripke structure is a structure $K = (W, R_{\mathbb{N}^{\mathcal{A}_B}}, \mathcal{V})$ 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 constant holds. $\mathcal{A}_B$ is a finite set of basic labels (called*

[9] The definition is standard, but several definitions of Kripke structures exist.

basic actions), $\mathbb{N}^{\mathcal{A}_B}$ *is the set of multisets built with the elements of* $\mathcal{A}_B$*, and* $R_{\mathbb{N}^{\mathcal{A}_B}} : \mathbb{N}^{\mathcal{A}_B} \to 2^{W \times W}$ *is a function returning for each multiset a set of pairs of worlds (intuitively* $R_{\mathbb{N}^{\mathcal{A}_B}}$ *gives a relation on the worlds for each multiset label).*

The rooted trees and the guarded rooted trees are defined as in sections 3.1 and 4 respectively. We use the notation T_n to denote the subtree of T with root in the node n of T.

Definition 6.2 (Simulation for rooted trees). *We say that a rooted tree* $T = (N, E, \mathcal{A})$ *is simulated by a labeled Kripke structure* $K = (W, R_{\mathbb{N}^{\mathcal{A}_B}}, \mathcal{V})$ *with respect to a state* r *of* K*, denoted* $T\mathcal{S}_r K$*, iff*

whenever $T\mathcal{S}_r K$ *then*
if $(r, \gamma, n) \in E$ *is an edge in* T *and* $\gamma \in \mathcal{A}$ *is a label*[10] *then*
$\exists w \in W$ *with* $(r, w) \in R_{\mathbb{N}^{\mathcal{A}_B}}(\gamma)$ *and* $T_n \mathcal{S}_w K$*.*

Definition 6.3 (Simulation for guarded rooted trees). *We say that a guarded rooted tree* $T = (N, E, \mathcal{A})$ *is simulated by a labeled Kripke structure* $K = (W, R_{\mathbb{N}^{\mathcal{A}_B}}, \mathcal{V})$ *with respect to a state* r *of* K*, denoted* $T\mathcal{S}_r K$*, iff*

$r \in \mathcal{V}(\phi)$ *where* ϕ *the guard of node* $r : \{\phi\}$ *of the tree* T*, and*
whenever $T\mathcal{S}_r K$ *then*
if $(r, \gamma, n) \in E$ *is an edge in* T*,* $\gamma \in \mathcal{A}$ *is a label, and* φ *of* $n : \{\varphi\}$ *is the guard of node* n *then*
$\exists w \in W$ *with* $(r, w) \in R_{\mathbb{N}^{\mathcal{A}_B}}(\gamma)$ *and* $w \in \mathcal{V}(\varphi)$*, and* $T_n \mathcal{S}_w K$*.*

Definition 6.4 (Partial simulation). *We say that a guarded rooted tree* $T = (N, E, \mathcal{A})$ *is partially simulated by a labeled Kripke structure* $K = (W, R_{\mathbb{N}^{\mathcal{A}_B}}, \mathcal{V})$ *with respect to a state* r *of* K*, denoted* $T\mathcal{S}'_r K$*, iff* $\exists T_r$ *a subtree of* T *starting at node* r[11] *such that* $T_r \mathcal{S}_r K$*.*

We consider a slight variation of a Kripke structure which we call *normative structure* and usually denote by $K^{\mathcal{N}}$. The normative structure was defined in [PS07a] but not with this particular name, here we just restate the definition for the sake of presentation.

Definition 6.5 (Normative structure). *A normative structure is a normal labeled Kripke structure as in Definition 6.1 with the following extensions:*

- *The labels are multisets on a set of basic actions* $\mathcal{A}_B$

[10] Remember that the labels of the rooted trees are multisets.
[11] Which is the root node in our case.

- *There is a set P_c of special propositional constants O_a and $\mathcal{F}_b$ indexed by the basic actions of $\mathcal{A}_B$*
- *The transitions are deterministic; i.e. the function $R_{\mathbb{N}^{\mathcal{A}_B}}$ associates to each label a function now instead of a relation, therefor for each label from one world there is only one reachable world.*

We have given in [PS07a] the semantics of $\mathcal{CL}$ with the help of a translation function which translated each $\mathcal{CL}$ syntax into $\mathcal{C}\mu$ syntax (a variant of the μ-calculus [Koz83]). The semantics of the logic is given in a set theoretical way on a Kripke structure. We take the equivalent way of giving semantics in terms of satisfiability w.r.t. a model and a state. Our model is the normative structure $K^{\mathcal{N}}$.

$K^{\mathcal{N}}, r \models O(\alpha)$ iff $I_{\mathcal{CA}}(\alpha)\mathcal{S}_r K^{\mathcal{N}}$ and
$\forall n \neq r \in N^\alpha$ with $I_{\mathcal{CA}}(\alpha) = (N^\alpha, E^\alpha, \mathcal{A}^\alpha)$
$\forall a \in \mathcal{A}_B$ with $M^\gamma(a) \geq 1$ where $(p, \gamma, n) \in E^\alpha$, $p < n$ then
$n \in \mathcal{V}(O_a)$

$K^{\mathcal{N}}, r \models P(\alpha)$ iff $I_{\mathcal{CA}}(\alpha)\mathcal{S}_r K^{\mathcal{N}}$ and
$\forall n \neq r \in N^\alpha$ with $I_{\mathcal{CA}}(\alpha) = (N^\alpha, E^\alpha, \mathcal{A}^\alpha)$
$\forall a \in \mathcal{A}_B$ with $M^\gamma(a) \geq 1$ where $(p, \gamma, n) \in E^\alpha$, $p < n$ then
$n \in \mathcal{V}(\neg\mathcal{F}_a)$

$K^{\mathcal{N}}, r \models F(\alpha)$ iff whenever $I_{\mathcal{CA}}(\alpha)\mathcal{S}'_r K^{\mathcal{N}}$ then, considering $I_{\mathcal{CA}}(\alpha) = T$,
$\forall T_r$ a subtree s.t. $T_r \mathcal{S}_r K^{\mathcal{N}}$, and $\forall \sigma$ a branch in T_r
$\exists (n, \beta, n') \in \sigma$ an edge in $T_r = (N^{T_r}, E^{T_r}, \mathcal{A}^{T_r})$ s.t.
$\forall (n, \gamma, m) \in K^{\mathcal{N}}$ with $\beta <_\& \gamma$ then
$\forall a \in \mathcal{A}_B$ with $M^\beta(a) \geq 1$, $m \in \mathcal{V}(\mathcal{F}_a)$

We pause now for some comments on the semantics above. For the F modality we use partial simulation $\mathcal{S}'_r$ 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 *existance* of a node on each path in order to respect the intuition that $F(a \cdot b) = F(a) \vee [a]F(b)$, forbidding a sequence means forbidding some action on that sequence. The last lines of the semantics of F look for all the transitions of the normative structure from the choosen 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.

With the above semantics we have the following holding:

$$F(a) \Rightarrow F(a\&b) \quad (24)$$
$$F(a+b) \equiv F(a) \wedge F(b) \quad (25)$$
$$P(a+b) \equiv P(a) \wedge P(b) \quad (26)$$
$$F(a \cdot b) \equiv F(a) \vee [a]F(b) \quad (27)$$
$$P(a \cdot b) \equiv P(a) \wedge [a]P(b) \quad (28)$$

All these have to be proven. For equation (24) for example the proof has to follow the standard way that $\forall M$ a model of $F(a)$ we must prove that it is also a model of $F(a\&b)$, i.e. if $M \models F(a)$ then $M \models F(a\&b)$. We prove a generalisation of equation (24) where instead of a and $a\&b$ we have any concurrent compound actions α and β s.t. the second is more demanding than the first.

Proposition 6.1. *$F(\alpha) \Rightarrow F(\beta)$, $\forall \alpha, \beta \in \mathcal{A}_{\&}$ and $\alpha <_{\&} \beta$, iff $\forall M$ a normative structure s.t. $M \models F(\alpha)$ then $M \models F(\beta)$*

Proof: PROOF HERE □

Moreover, the following do not hold (and this follows the intuition drawn from practice):

$$F(a\&b) \not\Rightarrow F(a) \quad (29)$$
$$P(a\&b) \not\Rightarrow P(a) \quad (30)$$

$\mathcal{CL}$ does not allow expressions like $F(a) \wedge (\varphi \Rightarrow P(a))$ which are not valid, but which may be intuitive for the reader as (s)he may think of real examples where some action is declared forbidden and only in some exceptional cases it is permitted. 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 from a logical point of view is also more natural.

For the other operators of $\mathcal{CL}$ (the dynamic modality $[\cdot]$ and $\langle\cdot\rangle$ or the temporal modalities $\Box$, $\Diamond$, or U) the semantics is the usual one. 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 $[\alpha]$ it looks only at the nodes at the boundery of the tree (the leaf nodes). 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] (if we think instead of each transition to be green that the states are green).

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.

7 Conclusion

In this paper we have introduced a new algebraic structure for true concurrent actions. The algebra faithfully formalizes the properties of the actions used in the contract language introduced in [PS07a]. A natural question here is, why to include true concurrency in the context of electronic contracts? There are many reasons for choosing true concurrency instead of interleaving. First, it reflects more naturally what it is expressed in natural languages when writing a contract, where some obligations can be stated on actions occurring simultaneously. Second, even in cases where true concurrency is not really required, or whenever it is impossible to detect simultaneity –as in run-time monitoring of events– true concurrency provides a more concise representation of the contract to be analyzed. This is indeed the case in the following two kinds of analysis. (1) When model checking contracts, the state-space is dramatically reduced by using true concurrency instead of interleaving; this applies even in the presence of partial order reduction techniques. (2) In run-time monitoring (for instance to monitor that contract violations do not occur) the monitor (automaton) obtained is definitely smaller if it has labels containing concurrent actions than in the presence of interleaving.

Besides the concurrent operator, the new features comprise: the definition of negation over non-atomic actions, including a procedure to "push" the negation only to atomic actions; test actions; and an action canonical form. Moreover, we have provided an interpretation of the algebra terms into rooted trees. Though the trees are in theory potentially infinite branching due to action negation, for our practical purposes in the context of contract specification (and run-time monitoring), this does not cause any problem. Indeed, when using the negation $\overline{a}$ of an action a, we never need to generate, or test against, all possible actions different from a: it only suffices to identify that the action is not a. The tree schema we have presented is thus extremely useful in practice.

We have also introduced a conflict relation to determine when two actions cannot be performed concurrently. In practice, when writing a contract, we provide the conflict relationship by listing which actions are in conflict. In this way we are able to reason and detect possible inconsistencies and contradictions.

The decision to define a new action-based algebraic structure instead of using previous work was not driven by a mere capricious intellectual challenge. In what follows we discuss and contrast our approach with other related work, showing why they are not suitable to our needs.

7.1 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 neutral 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].

Our algebra has three major differences with respect to the above works (dictated mainly by our application to e-contracts): (1) it has no Kleene star, (2) it has a true concurrency operator &, and (3) it can model discrete quantities.

In the following we relate the research done in this paper to other works.

Q-algebra [CK07]: An algebraic structure called *Q-algebra* is presented in [CK07] which is similar to our $\mathcal{CA}$ algebra because it has the same three operators *choice*, *sequence*, and *concurrent composition*. Basically Q-algebra is two idempotent semirings (which authors call "constraint semirings") but no further analisis of the relations between the operators is given. There are no axioms of the algebra and not much intuitive explanations nor application examples. Q-algebras do not have action negation, nor the notion of conflict actions, nor tests. Our interpretation as rooted trees is more appealing for the semantics of our contract language than the Q-automata.

On the other hand the theory of [CK07] is questionable in itself as it is based on wrong notions. For example the authors restate the definition of a constraint semiring (or c-semiring [BMR97]) as a normal idempotent semiring with the additive operation a *binary* operation. This definition is in contradiction with the original definition of c-semirings [BMR97] where the feature of a c-semiring (different than the classical idempotent semirings) is

that the additive operator is applied on a set of elements (i.e. either to zero, one, two, or all elements of the domain).

We would like to refer here more to the work on c-semirings [BMR97] which has similar notions to ours as most of the results presented for c-semirings are just restatements and adaptations of the classical results from idempotent semirings theory. The novelty of c-semirings is in the treatment of the additive operation, as it is applied (not as in classical semirings on two elements) on (possibly infinite) sets of elements. The application of c-semirings is to model constraints and thus they adopt a desirable property that the identity element **1** of the multiplicative operation is also an absorption element for the additive operation. This is because addition of all elements in the domain is defined to be equal to **1** in a c-semiring. Anyway, in any semiring the addition of all the elements of the domain gives a absorption element, and in particular in our $\mathcal{CA}$ at the level of the basic action, the finite choice of all the basic actions of $\mathcal{A}_B$ is absorption element for the choice operator $+$.

mCRL2 [GMR$^+$07]: The language mCRL2 for the specification and analysis of distributed systems introduced in [GMR$^+$07] by Groote et al, is based on well-founded algebraic theories. With the exception of the action negation introduced in our algebra, the underlying action algebra of mCRL2 contains all the other features we need, including true concurrent actions, and more. For our purposes, however, it seems more natural to define a new algebra since we do not need all the expressive power of the mCRL2 action algebra, and we need a special action negation not present there. Moreover, we provided a semantics of actions over rooted trees to make the connection with the contract language $\mathcal{CL}$.

Dynamic Deontic Logic [Mey88]: Actions similar to ours have been investigated in the framework of a deontic logic of actions by Meyer [Mey88]. The differences are that the concurrency operator of Meyer is basically the intersection of sets (as in extensions of PDL) and he does not consider the non-idempotence of it (as we do). Moreover, Meyer's action negation is defined with respect to the universal relation.

In [Mey88], as well as in [HTK00], action negation is with respect to the universal relation which, for example for PDL [Pra76] 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.

Our action negation is more general than just negation of atomic actions and at the same time it does not involve the universal relation. This leads us to conjecture that PDL extended with our kind of action negation does not yield undecidability.

Meyer's approach of defining obligation (i.e. $O(\alpha) = [\overline{\alpha}]V$) can be found in some variant in [And58] as $O\varphi \equiv \Box(\neg\varphi \supset s)$ where s is a propositional constant which means *violation* (something bad). This relation is discussed also by Meyer.

SCCS [Mil83]: Synchronous CCS (introduced in [Mil83]) is a rather general calculus developed in the same style as CCS. It has many features that we needed but the main drawback is that we could not see the way to integrate it (or a similar variant) within our logical setting. Many of the main features of SCCS are also found in our algebra $\mathcal{CL}$. These are: *atomic actions* as building blocks of the calculus; a *product* combinator which combines concurrently two agents and the combination of the actions is the same as in our case (forming a set of basic actions); an *action combinator* which basically is constructing the agents sequencely from basic actions; the *nondeterministic summation* or our choice. Besides these there is also a *restriction* operator; an *idling* operator; and an *action morphism* operator. There is also a *recursion* combinator with a fix point behavior. The recursion operator has the purpose of modelling *persistent agents*. The semantics of the operators is given with derivation rules (like in CCS) and later the equational properties of the operators are analyzed in a way similar to ours. We can view the derivation trees associated to each agent as similar to our rooted trees.

We do not have the Kleene * for recursion but we have the μ fix point operator at the level of the language. SCCS considers infinite summation thus generating infinitely branching derivation trees which is not what we desired. Our sequence operator is more general and compositional as it can put in sequence any arbitrary two actions in contrast with SCCS operator which can append only a basic action to an agent. The concurrency product operator of SCCS is not infinitary contrary to ours. On the other hand the argument in Milner [Mil83] that it is not so realistic to have this behavior is pertaining.

Milner has an interesting notion of *inverse of a basic action* with respect to the concurrent composition of two actions, i.e. $\overline{a}a = 1$ in SCCS notation.

Statecharts [Har87]: We should relate our $\mathcal{CAT}$ algebra also with Harel's statecharts [Har87] because, as our algebra claims to model a kind of concurrent execution of actions, statecharts are one of the well known concurrency formalisms. The relation is not so obvious as in the case of Milner's SCCS as statecharts are oriented towards a graphical representation of the reactive system (Statecharts are one of the first *visual languages*).

The main features of the statecharts are: they are a state-based formalism (extending Finite State Machines) which from one state, the system can change state in respect with *events*, *conditions* on states, and also have a Mealy-like output modelling *actions*. The conditions can be viewed as our guards on actions: if in statecharts an action does not have a condition then in $\mathcal{CAT}$ the guard is just $\top$?. A first extension of the FSMs is that statecharts include techniques for *clustering of states* into a superstate and *refinement* of one state into substates. This gives the formalism modularity and a well-structured hierarchical representation of a system (being now able to *zoom-in* and *zoom-out* of the model).

A second class of features of statecharts in *orthogonality* which includes *concurrency* and *independence*. Statecharts, as well as Milner's SCCS or Pratt's *pomsets* are models on concurrency which do not take the interleaving view. Concurrency in statecharts models how a system can be in several (clustered) states at the same time and execute several actions at the same time (from several of the concurrent states). This is not far from our view; in our case we consider only one state from which several actions can be executed concurrently. We do not have the notion of refinement or clustering of states so we kind of encode this into one state with several propositional constants holding in each state. Orthogonality is a more graphical-friendly and with fewer states of giving a synchronous product of the FSMs corresponding to the several concurrent components.

It is clear that we cannot use statecharts for our purpose of using the concurrent actions inside the logical language $\mathcal{CL}$. On the other hand many interesting ideas can be taken from this formalism and more, our way of considering concurrent actions goes well with the ideas presented in statecharts, which gives us a degree of confidence in our formalism.

Esterel [BG92]: Esterel is a synchronous language introduced in [BG92] (the journal version). Esterel is not well suited to be introduced inside our $\mathcal{CL}$ language as the basic theory and semantics for our actions, but many interesting ideas can be found. Firstly, Esterel has an *incompatibility relation* over events the same as ours over actions. Esterel adopts the *synchrony hypothesis* which basically states that every interactions in the model are

instantaneous. We adopted for now this same simplified view for our actions (as our actions do not take time to execute; or equivalently, at each tick of the clock all possible actions at that step are executed). We plan to extend the actions with *parameters* where one parameter may be of type real numbers such that it will be interpreted as the duration of the action. This extension will give greater modelling power for our actions, as it is needed for modelling real-life contracts.

In the following we discuss models of concurrency which have as primary object of discourse *events*, (partial) orderings over events and other kinds of relations. All these research efforts were motivated by the need to model executions of parallel and distributed computation. We have no preference on the order in which we base our presentation of the different models. There is not much distinction between events and actions as is the case in Esterel.

Event structures [NPW79]: Event structures were introduced in [NPW79] as a model of concurrency based on events partially ordered by a *causal dependency* relation and with additional structure given by a *conflict* relation and an *enabling* relation. We base our presentation on [Win88]. Configurations (or computation states) are viewed as subsets of events (left-closed w.r.t. the causal dependency order) which for one event all the events it depends on are included. Note that parallel processes computations are modeled by causal independence between events. Event structures are based on the fundamental *axiom of finite causes* which basically states that any event depends on a finite number of events. We like to note that our interpretation of actions as trees mimic event structures with the $\cdot$ as a causal dependency relation and any node respects the axiom of finite causes.

The conflict relation $\#$ is similar to our conflict relation and to the one found in Esterel. The conflict relation is defined over events and its intuitive interpretation is to express how the occurrence of an event rules out the occurrence of another. More general event structures $(E, \#, \vdash)$ are obtained by relaxing the partial order to a *enabling* relation $\vdash$ with the intuition that now it is sufficient for an event to be enabled by a single chain of enabled events starting with an event e_0 which is enabled by *no* event.

Event structures are much related to **Petri nets [Pet73]**. We do not analyze here the extensive literature on Petri nets, but we try to relate to the basic concepts of Petri nets for modelling concurrency. A transition in a Petri net may be triggered by occurrence concurrently of a set of events and some set of conditions. Moreover, a transition consumes the set of (pre)conditions and inserts a new set of (post)conditions. It is shown how via the Mazuriewicz traces [Maz84] a safe Petri net is equivalent to a (prime) event structure whose

events correspond to occurrences of events in the net.

Trace theory [Maz88]: As we have seen, event structures and Mazuriewicz traces, and we will see that also pomsets have many notions in common. Trace theory is also based on events (actually traces of events which may be thought as strings for concurrent processes) with a partial order and a causal dependency relation. As the (binary) causal dependency relation is defined on the set (alphabet) of events there is the natural relation of *independence* of two events. In traces one can see the notion of interleaving[12] as with the independence relation one may constructs equivalence classes of traces (sets of strings of events): e.g. if $(a, b) \in I \Rightarrow ab \equiv ba$ where I is the independence relation and $\equiv$ is the equivalence relation defined as shown. It turns out that the algebra of traces has a nice isomorphic graphical formalism called *dependency graphs* which make visually explicit the ordering of events within traces. Trace theory also has a nice notion of individual and global *history* of processes.

Pomsets [Pra86]: Pomsets have been long advocated by Pratt [Pra86] and many of the initial theoretical results were published as [Gis84]. Our presentation here is also based on [Gai88]. Pomsets are multisets of actions with two partial orders: *causal* precedence and *temporal* precedence.[13] The theory of pomsets is among the first in concurrency theory to make a distinction between events and actions. Normally a multiset is $\mathbb{N}^A$ and assigns to each action of A a multiplicity from $\mathbb{N}$. In pomset theory they are more: E^A which assigns to each action of A a set of events from E, and more, events are ordered by the temporal partial order. Thus, an action may be executed several times and each execution of an action is an event.

Pomsets make the distinction between *simultaneous* events (which are incomparable by the temporal precedence) and *concurrent* events (which are incomparable by the causal precedence). Sequentiality is given by comparability of two events with respect to the causal precedence (which is the smallest of the two).

Another feature of the pomset theory is that it is independent of the *granularity of the atomicity*; i.e. events may be either atomic or may have a even more elaborated structure. Moreover, the view of time does not matter as event may occupy time points or time intervals with no difference to the

[12] Interleaving is not mentioned explicitly in the related literature.

[13] Causal precedence is included in the temporal precedence, so Gaifmans's presentation does not diverge from Pratt's presentation.

theory. There is also a large number of operations defined over pomsets (see [Pra86]), more than in the other theories we have seen.

EXTEND THE PART WITH mCRL2

Acknowledgements

We would like to thank Martin Steffen for the very fruitful discussions 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 interesting and relevant related work.

References

[And58] Alan Ross Anderson. A reduction of deontic logic to alethic modal logic. *Mind*, 67(1):100–103, 1958.

[BG92] Gérard Berry and Georges Gonthier. The Esterel synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, 1992.

[BMR97] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint satisfaction and optimization. *Journal of ACM*, 44(2):201–236, 1997.

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

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

[CK07] Tom Chothia and Jetty Kleijn. Q-automata: Modelling the resource usage of concurrent components. In *5th International Workshop On the Foundations of Coordination Languages and Software Architectures (FOCLASA'06)*, volume 175 of *Electronic Notes In Theoretical Computer Science*, pages 153–167. Elsevier Science Publishers B. V., 2007.

[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.

[dBdRR89] J. W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg, editors. *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, School/Workshop, Noordwijkerhout, The Netherlands, May 30 - June 3, 1988, Proceedings*, volume 354 of *Lecture Notes in Computer Science*. Springer, 1989.

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

[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.

[Gai88] Haim Gaifman. Modeling concurrency by partial orders and nonlinear transition systems. In de Bakker et al. [dBdRR89], pages 467–488.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Gis84] Jay L. Gischer. *Partial Orders and the Axiomatic Theory of Shuffle*. PhD thesis, CS, Stanford University, 1984.

[GMR+07] Jan Friso Groote, Aad Mathijssen, Michel Reniers, Yaroslav Usenko, and Muck van Weerdenburg. The formal specification language mCRL2. In *MMOSS'07*, number 06351 in Dagstuhl Seminar Proceedings, 2007.

[Har87] David Harel. Statecharts: A visual formulation for complex systems. *Science of Computer Programmings*, 8(3):231–274, 1987.

[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.

[Maz84] Antoni W. Mazurkiewicz. Traces, histories, graphs: Instances of a process monoid. In Michal Chytil and Václav Koubek, editors, *MFCS*, volume 176 of *Lecture Notes in Computer Science*, pages 115–133. Springer, 1984.

[Maz88] Antoni W. Mazurkiewicz. Basic notions of trace theory. In de Bakker et al. [dBdRR89], pages 285–363.

[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.

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

[NPW79] Mogens Nielsen, Gordon D. Plotkin, and Glynn Winskel. Petri nets, event structures and domains. In Gilles Kahn, editor, *Semantics of Concurrent Computation*, volume 70 of *Lecture Notes in Computer Science*, pages 266–284. Springer, 1979.

[Pet73] C. A. Petri. Concepts of net theory. In *MFCS*, pages 137–146, 1973.

[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.

[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. A formal language for electronic contracts. In *FMOODS'07*, volume 4468 of *LNCS*, pages 174–189. Springer, 2007.

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

[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.

[Win88] Glynn Winskel. An introduction to event structures. In de Bakker et al. [dBdRR89], pages 364–397.