An Algebraic Structure for Concurrent Actions*

Cristian Prisacariu

Department of Informatics – University of Oslo,
P.O. Box 1080 Blindern, N-0316 Oslo, Norway.
cristi@ifi.uio.no

Abstract

In [2] we have provided a formal language for specifying contracts, which allows to write (conditional) obligations, permissions and prohibitions of the different contract signatories, based on the so-called *ought-to-do* approach. In such an approach the above normative notions are specified over (names of human) *actions*, as for example "The client is obliged to pay after each delivery". There, we have 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 presented in [2]. Though the algebraic structure we define is somehow similar to Kleene algebra with tests [1], there are substantial differences due mainly to our application domain. A first difference is that we do not include the Kleene star as it is not needed in our context. A second difference is that we introduce an operator in the algebra to model true concurrency. The main contributions of the paper are: (1) A formalization of concurrent actions; (2) The introduction of a different kind of action negation; (3) A restricted notion of resource-awareness; and (4) A standard interpretation of the algebra over specially defined rooted trees.

The *algebra of concurrent actions and tests* ($\mathcal{CAT}$) that we present in this abstract is formed of an algebraic structure $\mathcal{CA} = (\mathcal{A}, +, \cdot, \&, \mathbf{0}, \mathbf{1})$ which defines the concurrent actions, and a Boolean algebra which defines the tests. Special care is taken when combining actions and tests under the different operators.

The algebraic structure $\mathcal{CA}$ is defined by a carrier set of elements (which we call *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*. 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. The constant function symbols of the finite set $\mathcal{A}_B \subseteq \mathcal{A}$ are called basic (atomic) actions. The special elements $\mathbf{1}$ and $\mathbf{0}$ are also constant function symbols. The set of basic actions is called the *generator set* of the algebra. In Table 1 we collect the axioms that define the structure $\mathcal{CA}$.

We want to have a resource-aware algebra similarly to what has been done for linear logic. Therefore we do not allow the idempotence property for the $\&$ operator ($a \& a \neq a$). As an example, if α represents the action of paying 100$

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

(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 $length(\alpha) = length(\alpha') = n$
(9)	$(\alpha + \beta) \cdot \gamma = \alpha \cdot \gamma + \beta \cdot \gamma$		

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

then paying 200$ would be represented as $\alpha\&\alpha$. Note that we can represent only discrete quantities with this approach. We consider a *conflict relation* over the set of basic actions $\mathcal{A}_B$ (denote by $\#_{\mathcal{C}}$) defined as: $a \#_{\mathcal{C}} b \stackrel{def}{\iff} a\&b = \mathbf{0}$. The intuition of the conflict relation is that if two actions are in conflict then the actions cannot be executed concurrently.

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. 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. 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 ?.

We give the standard interpretation of the actions of $\mathcal{A}$ by defining a homomorphism $I_{\mathcal{CAT}}$ which takes an action of the $\mathcal{CAT}$ algebra and returns a special guarded rooted tree preserving the structure of the action given by the constructors. A guarded rooted tree has labels (representing basic actions) on edges and tests as the types of the nodes. We define special operators on these trees: $\cup$ join, $\hat{}$ concatenation, and $\|$ concurrent join. In order to have the same behavior of $\mathbf{1}$ and $\mathbf{0}$ from $\mathcal{CAT}$ under the interpretation as trees we give a special procedure for *pruning* the trees.

For actions α defined with the operators $+$, $\cdot$, $\&$, and tests we have a canonical form denoted $\alpha!$ and defined as: $\alpha! = +_{\rho \in R}\ \rho \cdot \alpha'!$, where R contains either basic actions, concurrent actions, or tests, and α' is a compound action in canonical form. The action negation is denoted by $\overline{\alpha}$ and is defined as: $\overline{\alpha} = \overline{+_{\rho \in R} \rho \cdot \alpha'} = +_{b \in \overline{R}} b\ +\ +_{\rho \in R} \rho \cdot \overline{\alpha'}$, where ρ and α' are as before. The set $\overline{R}$ is defined to contain: $\{(\neg\phi)? \mid \phi \in R\} \cup \{\alpha \mid \alpha \in \mathcal{A}_\&,$ and $\forall \beta \in R, \beta \not<_\& \alpha\}$ where $\mathcal{A}_\&$ contains concurrent actions generated only by means of &, and $<_\&$ is a strict partial order which basically compares two actions to see which contains the other with respect to the & operator. Note that because & is not idempotent the set $\overline{R}$ becomes infinite (thus having infinite branching in the associated tree). We overcome this problem by defining *action schemas* and *tree schemas*.

In conclusion we mention some works which are close related to our work. J.J.Meyer '88 investigates algebraic properies of the actions he has in its Dynamic Deontic Logic. D.Kozen's extensive work on Kleene algebras forms a basis for our algebra. Our work goes well with Pratt's work on pomsets for true concurrency.

A nice introduction to rooted trees can be found in the work of M.Hennessy on algebraic theory of processes.

Acknowledgements

All the results of the paper have been obtained jointly with Gerardo Schneider.

References

1. D. Kozen. Kleene algebra with tests. *TOPLAS'97*, 19(3):427–443, 1997.
2. C. Prisacariu and G. Schneider. A formal language for electronic contracts. In M. Bonsangue and E. B. Johnsen, editors, *FMOODS'07*, volume 4468 of *LNCS*, pages 174–189. Springer, 2007.