Extending Kleene Algebra with Synchrony: Completeness and Decidability[⋆] (extended abstract)[⋆⋆]

Cristian Prisacariu

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

1 Motivation

The work reported here investigates the introduction of synchrony into Kleene algebra. The resulting algebraic structure is called *synchronous Kleene algebra*. Models are given in terms of regular sets of concurrent strings and finite automata accepting concurrent strings. The extension of synchronous Kleene algebra with Boolean tests is presented together with models on regular sets of guarded concurrent strings and the associated automata on guarded concurrent strings. Completeness w.r.t. the standard interpretations is given in each case. Decidability follows from completeness. A comparison with Mazurkiewicz traces is made which yields their incomparability with the synchronous Kleene algebra (one cannot simulate the other). Applications to respectively deontic logic of actions and to propositional dynamic logic with synchrony are sketched.

Kleene algebra is a formalism used to represent and reason about programs, and it formalizes axiomatically the structures of regular expressions and finite automata (see work of J.H. Conway and D. Kozen). Kleene algebra with tests combines Kleene algebra with a Boolean algebra and can encode the propositional Hoare logic, therefore it can express while programs. In one form or another, Kleene algebras appear in various formalisms in computer science: relation algebras, logics of programs and in particular propositional dynamic logic (PDL), regular expressions and formal language theory.

Synchrony is a model for concurrency which was introduced in the process algebra community in R. Milner's SCCS, but detaches form the general interleaving approach. On the other hand it is a concept which does not belong to the partial order model of true concurrency either. The synchrony concept proves highly expressive and robust; SCCS can represent CCS (i.e. asynchrony) as a subcalculus, and a great number of synchronizing operators can be defined in terms of the basic SCCS-Meije operators.

The notion of *synchrony* has different meanings in different areas of computer science. Here we take the distinction between *synchrony* and *asynchrony* as presented in

[⋆] Partially supported by the Nordunet3 project "COSoDIS – Contract-Oriented Software Development for Internet Services" (http://www.ifi.uio.no/cosodis/).
[⋆⋆] One may like to check details in the technical report: C. Prisacariu, "Extending kleene algebra with synchrony – technicalities", Dept. Informatics, Univ. Oslo, 2008.

the SCCS calculus and later implemented in e.g. the Esterel (Meije) synchronous programming language. We understand *asynchrony* as when two concurrent systems execute at indeterminate relative speeds (i.e. their actions may have different noncorelated durations); whereas in the *synchrony* model each of the two concurrent systems execute instantaneously a single action at each time instant. The SCCS concurrency operator $\times$ over processes is different from the classical $||$ of CCS.

The perfectly synchronous concurrency model takes the assumption that time is discrete and that basic actions are instantaneous (i.e. take zero time and represent the time step). Moreover, at each time step all possible actions are performed. The reasoning is governed by the assumption of a global clock which provides the time unit for all the actors in the system. Note that for practical purposes this is a rather strong assumption which offends the popular relativistic view from process algebras. On the other hand the mathematical framework of the synchronous calculus is much cleaner and more expressive than the asynchronous model, and the experience of the Esterel implementation and use in industry contradict the general believe.

The motivation for adding synchrony to Kleene algebra spawns from the need to reason about actions (where Kleene algebra is the equational tool of choice in conjunction with PDL) that can be executed in a truly concurrent fashion. On the one hand we do not need such a powerful concurrency model like the ones based on partial orders; on the other hand the low level interleaving model is not expressive enough. The synchrony model has appealing equational representation and thus it is natural to integrate into the Kleene algebra. Moreover, the reasoning power and expressivity that synchrony offers is enough for the applications listed below.

We have successfully used synchronous Kleene algebra in the context of deontic logic of actions, and secondly we used the extension of synchronous Kleene algebra with tests in the context of PDL with synchrony. A third application that we currently work on is to give a semantics for Java threads. More general, wherever one uses Hoare logic to reason about programs one can use the more powerful Kleene algebra with tests which has also tool support (the KAT-ML prover). Moreover, one may safely choose the synchronous Kleene algebra with tests where in addition reasoning about concurrent executions is needed (similar to some extent to the current work of C.A.R. Hoare). In these contexts (synchronous) Kleene algebra proves more powerful and general than classical logical formalisms.

2 Results

Synchronous Kleene algebra ($\mathcal{SKA}$) adds to the Kleene algebra the $\times$ operator of SCCS. $\mathcal{SKA}$ is a σ-algebra with signature $\sigma = \{+, \cdot, \times, ^*, \mathbf{0}, \mathbf{1}, \mathcal{A}_B\}$ which gives the action operators and the basic actions $\mathcal{A}_B$. The operators of $\mathcal{SKA}$ are defined over a carrier set of *compound actions* denoted $\mathcal{A}$. The non-constant functions of σ are: "$+$" for *choice* of two actions, "$\cdot$" for *sequence* of two actions (or concatenation), "$\times$" for *concurrent composition* of two actions, and "*" to model *recursive execution* of one action.

We give the *standard interpretation* of the actions by defining a *homomorphism* $\hat{I}_{\mathcal{SKA}}$ which takes any action of the $\mathcal{SKA}$ algebra into a corresponding *regular concurrent set* and preserves the structure of the actions given by the operators. A *concurrent* *set* is a subset of strings from $\mathcal{P}(\mathcal{A}_B)^*$. The important construction on concurrent sets is the one corresponding to $\times$: $A \times B \triangleq \{u \times v \mid u \in A, v \in B\}$, where $u, v \in \mathcal{P}(\mathcal{A}_B)^*$ are concurrent strings and $u \times v$ is defined as (where $x_i, y_j \in \mathcal{P}(\mathcal{A}_B)$):

$$u \times v \triangleq (x_1 \cup y_1)(x_2 \cup y_2) \dots (x_n \cup y_n) y_{n+1} \dots y_m.$$

Theorem 1 (completeness). *For any actions $\alpha, \beta \in \mathcal{A}$ then $\alpha = \beta$ is a theorem of $\mathcal{SKA}$ iff the regular concurrent sets $\hat{I}_{\mathcal{SKA}}(\alpha)$ and $\hat{I}_{\mathcal{SKA}}(\beta)$ are the same.*

The proof of completeness follows the ideas of D. Kozen only that we use a combinatorial argument. We use the translation of actions into automata that accept regular concurrent sets (and need to prove an equivalent of Kleene's representation theorem). The main ideas are that the alphabet of the automata is $\mathcal{P}(\mathcal{A}_B)$ and that we require a special product construction. Decidability in P-time follows then naturally from completeness and decidability of the equivalence problem for regular concurrent sets.

Synchronous Kleene algebra with tests $\mathcal{SKAT} = (\mathcal{A}, \mathcal{A}^?, +, \cdot, \times, ^*, \neg, \mathbf{0}, \mathbf{1})$ is an order sorted algebraic structure which combines $\mathcal{SKA}$ with a Boolean algebra in a special way. The structures $(\mathcal{A}^?, +, \cdot, \neg, \mathbf{0}, \mathbf{1})$ and $(\mathcal{A}^?, +, \times, \neg, \mathbf{0}, \mathbf{1})$ are Boolean algebras and the Boolean negation operator $\neg$ is defined only on Boolean elements (called *tests*) of $\mathcal{A}^? \subseteq \mathcal{A}$. The intuition behind tests is that for an action $\phi \cdot \alpha$ it is the case that action α can be performed only if the test ϕ succeeds.

Actions of $\mathcal{SKAT}$ are interpreted over what we call regular sets of *guarded concurrent strings* (an extension of guarded strings). Special two levels finite automata are defined which accept these regular sets; fusion product and concurrent composition are particular operations on these automata.

We compared the synchrony notion of $\mathcal{SKA}$ with Mazurkiewicz traces. Basically the Mazurkiewicz traces have defined a *global* and *partial* independence relation. If we take the similar view in $\mathcal{SKA}$ we need a *local* and *total* independence relation. The locality comes from the perfect synchrony model we adopted, where all the concurrent actions are executed at each tick of a universal clock. The totality comes from our view of concurrent basic actions as forming a set.

We also related to the *shuffle* operator over regular languages which has been used to model concurrency, with a position between the interleaving approach and the partial orders approach. Shuffle is a generalization of interleaving but it does not take into considerration any other relation on the actions(events) which it interleaves. If we were to ignore the branching information in our actions then we can view $\times$ as an *ordered shuffle*. The shuffling of two sequences of actions in $\mathcal{SKA}$ walks step by step (on the $\cdot$ operation) and shuffles the basic actions found (locally).

Our work is also related to Q-algebra which is a two idempotent semirings structure. The difference is in our introduction of the notion of synchrony and the relation of the $\times$ operator with the sequence operator. Also related to our algebraic structure is the language mCRL2. This is a too complex formalism (and tool set) to be naturally incorporated in the applications to the logics that we mentioned in the beginning.