Synchronous Kleene Algebra[☆]

Cristian Prisacariu[1]

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

Abstract

The work presented here investigates the combination of Kleene algebra with the synchrony model of concurrency from Milner's SCCS calculus. The resulting algebraic structure is called *synchronous Kleene algebra*. Models are given in terms of sets of synchronous strings and finite automata accepting synchronous strings. The extension of synchronous Kleene algebra with Boolean tests is presented together with models on sets of guarded synchronous strings and the associated automata on guarded synchronous strings. Completeness w.r.t. the standard interpretations is given for each of the two new formalisms. Decidability follows from completeness. Kleene algebra with synchrony should be included in the class of true concurrency models. In this direction, a comparison with Mazurkiewicz traces is made which yields their incomparability with synchronous Kleene algebras (one cannot simulate the other). On the other hand, we isolate a class of pomsets which captures exactly synchronous Kleene algebras. We present an application to Hoare-like reasoning about parallel programs in the style of synchrony.

Key words:
Universal algebra, Kleene algebra, Boolean tests, synchrony, SCCS calculus, concurrency models, automata theory, completeness, Hoare logic

1. Introduction

Kleene algebra is a formalism used to represent and reason about programs. *Kleene algebra with tests* combines Kleene algebra with a Boolean algebra; it can express while programs [1] and can encode propositional Hoare logic using a Horn-style inference system. In one form or another, Kleene algebras appear in various formalisms in computer science: relation algebras, logics of programs, in particular, Propositional Dynamic Logic [2, 3], and regular expressions and formal language theory [4].

[☆]An extended abstract of this article was first presented in the 20^{th} Nordic Workshop on Programming Theory (NWPT'08) in Tallinn, Estonia, in November 2008.

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

In this paper we investigate the extension of Kleene algebra with a particular notion of concurrent actions which adopts the synchrony model. *Synchrony* is a model of concurrency which was introduced in the process algebra community in R. Milner's SCCS calculus [5] but which detaches from the general interleaving approach. Synchrony is a concept which belongs to the partial orders model of concurrency [6, 7, 8]. We see in Section 4 how synchrony as defined here compares to Mazurkiewicz traces [6] and pomsets [8]. The synchrony concept proves highly expressive and robust; SCCS can represent CCS (i.e., asynchrony) as a sub-calculus, and a great number of synchronizing operators can be defined in terms of the basic SCCS-Meije operators [9]. Meije is the calculus at the basis of the Esterel synchronous programming language [10]; Meije and SCCS operators are interdefinable.

The motivation for adding synchrony to Kleene algebra spawns from the need to represent and reason about actions which can be performed "*at the same time*". We view this notion as being closer to the models of concurrency based on partial orders than to the ones based on interleaving. For reasoning about actions we choose the established equational formalism of Kleene algebra. For a faithful representation of the notion of "*at the same time*" in an equational setting the synchrony model is the most appealing. We do not need such powerful concurrency models like the ones based on partial orders; on the other hand, the low level interleaving model is not well suited for (abstract) reasoning about actions done at the same time (and related properties). The synchrony model has an equational representation and thus it is easy to integrate into Kleene algebra. Moreover, the reasoning power and expressivity that synchrony offers is enough for the applications listed in Section 1.1.

This paper defines two algebraic structures obtained from the combination of Kleene algebra with synchrony, and investigates theoretical aspects of them. Section 2 contains the *synchronous Kleene algebra* (*SKA*), which is Kleene algebra extended with a synchrony combinator for actions. The main difficulties in the axiomatization of *SKA* come from the definition of the synchrony combinator and its relations with the other operators. For the definition of standard models for *SKA* we introduce *sets of synchronous strings* and give the relation between the actions and the models. We prove completeness of the axiomatization w.r.t. the standard models. From completeness we get the decidability of the equality between actions. To prove completeness we need to define a new kind of automata which recognize sets of synchronous strings. For these automata we prove standard results which we need in the completeness proof. The most important is the equivalent of Kleene's theorem which shows how to build an automaton for an action of *SKA* that accepts exactly the set of synchronous strings interpreting the action.

Besides the pure theoretical stimulation, the *SKA* formalism finds application in deontic logic over synchronous actions and to propositional dynamic logic over synchronous actions. More precisely, the *-free actions of *SKA* (and related results) are used in giving a direct semantics to the action-based contract-specification language $\mathcal{CL}$ [11]. Results for these logics may be found in [12].

Other applications of *SKA* can be found among the various places where

Kleene algebra is used (some of which we state in Section 1.1) and where is involved a notion of concurrency for which the synchrony model of *SKA* is expressive enough. To evaluate the expressiveness of *SKA* we give in Section 4 comparisons with related concurrency models like Mazurkiewicz traces, pomsets, and concurrent Kleene algebras. The concluding section contains more related work and open problems.

Section 3 contains the extension of synchronous Kleene algebra with Boolean *tests* (*SKAT*) which follows the methodology of extending Kleene algebra with tests of [13]. Mainly, we extend *SKA* with a Boolean algebra defining the tests (called *guards* in the models). At the axiomatization level there are no considerable novelties w.r.t. *SKA*. More difficult is to find standard models; these we define as *sets of guarded synchronous strings*. In the completeness proof of *SKAT* we use again an automata theoretic argument and thus we define automata to accept guarded synchronous strings. Here the operations over the automata are not standard, and care needs to be taken when defining the fusion product and the synchrony product. Using these, the equivalent of Kleene's theorem leads the way to proving the completeness, and thus the decidability.

One of the standard applications of Kleene algebra with tests (*KAT*) is to reason about programs in a more general way than with standard propositional Hoare logic. In the same line *SKAT* can be used to reason about concurrent programs with shared variables in the style of Owicki and Gries [14]. We present this application in Section 3.4.

The rest of this introductory section gives background material on Kleene algebra and synchrony, and can safely be skipped by an expert reader.

1.1. Applications

One application of synchronous Kleene algebras is in the context of the deontic logic of actions [15, 16] (underlying the semantics of the contract language $\mathcal{CL}$ [12]) and propositional dynamic logic with synchronous actions. These applications to deontic and dynamic logics are not part of this paper. The second application, in Section 3.4, presents *SKAT* as an alternative to Hoare logic for reasoning about parallel programs with shared variables in the synchrony style.

More generally, wherever one uses Hoare logic to reason about programs one can use the more powerful Kleene algebra with tests (the KAT-ML prover [17, 1] may be used to reason about programs in the style of Kleene algebra with tests). Similarly, one may safely choose *SKAT* when reasoning about concurrent executions is needed (similar to some extent with the current work of C.A.R. Hoare [18]). In these contexts (synchronous) Kleene algebra proves more powerful and more general than classical logical formalisms.

As other applications, we envisage the use of synchronous Kleene algebra to give semantics for Java threads and to give semantics to an extension of propositional dynamic logic (PDL) with synchrony. (In the same way as *KAT* is the underlying formalism for the programs (actions) of PDL, *SKAT* would be underlying the synchronous programs of this extension.) This would be an alternative to the PDL$^\cap$ [19] or concurrent PDL [20].

(1)	$\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$	(10)	$\mathbf{1} + \alpha \cdot \alpha^* \leq \alpha^*$
(2)	$\alpha + \beta = \beta + \alpha$	(11)	$\mathbf{1} + \alpha^* \cdot \alpha \leq \alpha^*$
(3)	$\alpha + \mathbf{0} = \mathbf{0} + \alpha = \alpha$	(12)	$\beta + \alpha \cdot \gamma \leq \gamma \ \rightarrow \ \alpha^* \cdot \beta \leq \gamma$
(4)	$\alpha + \alpha = \alpha$	(13)	$\beta + \gamma \cdot \alpha \leq \gamma \ \rightarrow \ \beta \cdot \alpha^* \leq \gamma$
(5)	$\alpha \cdot (\beta \cdot \gamma) = (\alpha \cdot \beta) \cdot \gamma$		
(6)	$\alpha \cdot \mathbf{1} = \mathbf{1} \cdot \alpha = \alpha$		
(7)	$\alpha \cdot \mathbf{0} = \mathbf{0} \cdot \alpha = \mathbf{0}$		
(8)	$\alpha \cdot (\beta + \gamma) = \alpha \cdot \beta + \alpha \cdot \gamma$		
(9)	$(\alpha + \beta) \cdot \gamma = \alpha \cdot \gamma + \beta \cdot \gamma$		

Table 1: Axioms of Kleene algebra

1.2. Kleene algebras

Kleene algebra (*KA*) was named after S.C. Kleene who in the fifties studied regular expressions and finite automata [21]. Kleene algebra formalizes axiomatically these structures. Further developments on the algebraic theory of *KA* were done by J.H. Conway [22]. For references and an introduction to Kleene algebra see the extensive work of D. Kozen [23, 24, 13]. Completeness of the axiomatization of *KA* was studied in [25, 26], complexity in [27], and applications to concurrency control, static analysis and compiler optimization, or pointer arithmetics in [28, 29, 30, 31]. Some variants of *KA* include the notion of *tests* [13], and others add some form of types [32].

Definition 1.1. *An* idempotent semiring *is an algebraic structure* $(\mathcal{A}, +, \cdot, \mathbf{0}, \mathbf{1})$ *that respects axioms (1)-(9) of Table 1. A* Kleene algebra $(\mathcal{A}, +, \cdot, ^*, \mathbf{0}, \mathbf{1})$ *is an idempotent semiring with one extra unary (postfix) operation* **, which respects the extra axioms (10)-(13) of Table 1. We understand the operations as representing respectively* nondeterministic choice*,* sequence*, and* iteration*. Henceforth we denote elements of* $\mathcal{A}$ *by* α, β, γ*, and call them* (compound) actions*. The constants* $\mathbf{0}$ *and* $\mathbf{1}$ *are sometimes called the* fail *action respectively the* skip *action. For an idempotent semiring the* natural order $\leq$ *is defined as:*

$$\alpha \leq \beta \stackrel{\Delta}{=} \alpha + \beta = \beta;$$

and in this paper we usually say that "β *is* preferable *to* α*".*

An intuitive understanding of the natural order of a semiring is that $\leq$ states that the left operand has less behavior than the right operand, or in other words, the right operand specifies behavior which includes all the behavior specified by the left operand (and possibly more).

Remarks: It is easy to check that $\leq$ is a partial order and that it forms a semilattice with least element $\mathbf{0}$ and with $\alpha + \beta$ the least upper bound of α and β. Moreover, the three operators are monotone w.r.t. $\leq$.

Notation: Fix a set of *basic (or atomic) actions* $\mathcal{A}_B$ (henceforth denoted by $a, b, c \in \mathcal{A}_B$). Consider the corresponding term algebra $T_{KA}(\mathcal{A}_B)$ that is finitely generated from the generator set $\mathcal{A}_B \cup \{\mathbf{0}, \mathbf{1}\}$. We use T_{KA} whenever

$\mathcal{A}_B$ is understood from context. The syntactic terms of T_{KA} (the *actions*) can be seen as generated by the grammar:

$$\alpha \quad ::= \quad a \mid \mathbf{0} \mid \mathbf{1} \mid \alpha + \alpha \mid \alpha \cdot \alpha \mid \alpha^*$$

where $a \in \mathcal{A}_B$. An acquainted reader may remark that $T_{KA}(\mathcal{A}_B)$ corresponds to the set of regular expressions over the alphabet $\mathcal{A}_B$.

The axioms (1)-(4) define the choice operator $+$ to be associative, commutative, with neutral element $\mathbf{0}$, and idempotent. Axioms (5)-(7) define the sequence operator $\cdot$ to be associative, with neutral element $\mathbf{1}$, and with annihilator $\mathbf{0}$ both on the left and right side. Axioms (8) and (9) give the distributivity of $\cdot$ over $+$.

The equations (10) and (11) and equational implications (12) and (13) are the standard axiomatization of * [25, 26] which say that $\alpha^* \cdot \beta$ is the least solution w.r.t. the preference relation $\leq$ for the equation $\beta + \alpha \cdot X \leq X$ (and dually $\beta \cdot \alpha^*$ is the least solution to the equation $\beta + X \cdot \alpha \leq X$).

Examples of Kleene algebras: Consider, in language theory, Σ^* the set of all finite words over the alphabet Σ [33]. The powerset $\mathcal{P}(\Sigma^*)$ (i.e., the set of all languages) with the standard operations of union, concatenation, and Kleene star over languages forms a Kleene algebra.

For a second example consider the set of all *binary relations* over a set X. The powerset of $X \times X$ with the standard empty relation (for $\mathbf{0}$), identity relation (for $\mathbf{1}$), union of relations, relational composition, and the transitive and reflexive closure of a relation (for *) forms a Kleene algebra. This algebra is used in the semantics of logics of programs, like Propositional Dynamic Logic [34, 19].

As a last example consider the less known *min,+ algebra* (also called the tropical algebra) which is useful in shortest path algorithms on graphs [35]. The operations are defined over the domain $\mathbb{R}_+ \cup \{\infty\}$. The $+$ operation from Kleene algebra is defined as the *min* operation on reals giving the minimum of two elements under the natural order on $\mathbb{R}_+ \cup \{\infty\}$ where ∞ is always the greatest element. The operation $\cdot$ is interpreted as the standard arithmetic $+$ on $\mathbb{R}_+ \cup \{\infty\}$. The two constants $\mathbf{0}$ and $\mathbf{1}$ are interpreted respectively as ∞ and 0. The * operation is surprisingly defined as $x^* = 0$.

Note that for the first two examples above the preference relation $\leq$ is defined to be set inclusion $\subseteq$ whereas for the last example it is the reverse of the natural order on reals.

1.3. Synchrony

The notion of *synchrony* has different meanings in different areas of computer science. Here we take the distinction between *synchrony* and *asynchrony* as presented in the SCCS calculus of [5] and later implemented in, e.g., the Esterel synchronous programming language [10, 36]. We understand *asynchrony* as the execution of two concurrent systems at independent relative speeds (i.e., their actions may have different non-correlated durations), whereas in the *synchrony*

model each of the two concurrent systems execute instantaneously a single action at each time instant.

The *synchrony model* takes the assumption that time is discrete and that basic actions are instantaneous. Moreover, at each time step, all possible actions are performed, i.e., the system is considered *eager* and *active* (idling is not possible). Synchrony assumes a global clock which provides the time unit for all the actors in the system. For practical purposes this is a rather strong assumption which is in contrast with the popular view from process algebras [37, 38]. On the other hand, the equational framework of the synchrony model is much cleaner and more general than the asynchronous interleaving model; the well known CCS calculus [37] is just a sub-calculus of the asynchronous version of SCCS (named ASCCS) [5]. Moreover, the experience of the Esterel implementation and use in industry contradict the general belief.

Regarding expressivity, the work of [9] establishes the relative completeness of expressiveness of the SCCS and Meije languages (i.e., they are equally expressive). Meije was the core language of the Esterel synchronous programming language, which is now widely used in industry.

SCCS introduces a synchronous composition operator $\times$ over processes which is different from the well known $\|$ of CCS (actually $\|$ can be defined in terms of $\times$). SCCS keeps the process algebra style of giving meaning to processes using structural operational semantics. The operational semantics of $\times$ is:

$$\frac{P \stackrel{a}{\rightarrow} P' \qquad Q \stackrel{b}{\rightarrow} Q'}{P \times Q \stackrel{a \times b}{\rightarrow} P' \times Q'}$$

In this paper we do not use structural operational semantics, but take an algebraic equational view in the style of Kleene algebras.

2. Kleene Algebra with Synchrony

We add to the standard Kleene algebra of Section 1.2 an operator to model *concurrency* similar to the synchronous composition of SCCS presented in Section 1.3. We call the resulting algebra *synchronous Kleene algebra* and abbreviate it *SKA*. The *SKA* algebra has the following particularities:

1. It formalizes a notion of *concurrent actions* based on the synchrony model.
2. It has a standard interpretation of the actions as *sets of synchronous strings*. The actions can also be represented as special finite automata which accept the same sets of synchronous strings that form the models of the actions.
3. It incorporates a notion of *conflicting actions*.

This section (as well as the next) is concerned with the theoretical investigations of the new algebra. The general motivations are given in the introduction and conclusion as well as through the examples of applications from the end of

All axioms of Kleene algebra from Table 1

(14) $\alpha \times (\beta \times \gamma) = (\alpha \times \beta) \times \gamma$

(15) $\alpha \times \beta = \beta \times \alpha$

(16) $\alpha \times \mathbf{1} = \mathbf{1} \times \alpha = \alpha$

(17) $\alpha \times \mathbf{0} = \mathbf{0} \times \alpha = \mathbf{0}$

(18) $a \times a = a \quad \forall a \in \mathcal{A}_B$

(19) $\alpha \times (\beta + \gamma) = \alpha \times \beta + \alpha \times \gamma$

(20) $(\alpha + \beta) \times \gamma = \alpha \times \gamma + \beta \times \gamma$

(21) $(\alpha_\times \cdot \alpha) \times (\beta_\times \cdot \beta) = (\alpha_\times \times \beta_\times) \cdot (\alpha \times \beta)$, $\forall \alpha_\times, \beta_\times \in \mathcal{A}_B^\times$

Table 2: Axioms of *SKA*

each section, and through the comparisons that we do in Section 4. Occasionally we give short intuitions for the particular notions presented and suggest applications.

The investigation of synchronous actions in an algebraic setting implies that one should consider them in the most general (and abstract) manner. Particular views of the actions can be as human-like actions from legal contracts, as instructions in a programming language, or as parallel executing processes.

2.1. Syntax and axiomatization

Definition 2.1 (Synchronous Kleene Algebra). *A* synchronous Kleene algebra *(SKA) is a structure* $(\mathcal{A}, +, \cdot, \times, ^*, \mathbf{0}, \mathbf{1}, \mathcal{A}_B)$ *obtained from a Kleene algebra by adding a "*$\times$*" operation for* synchronous composition *of two actions. The new operation* $\times$ *respects the axioms (14)-(21) of Table 2.*

Notation: Consider the set $\mathcal{A}_B^\times \subset \mathcal{A}$ to be the set $\mathcal{A}_B$ closed under application of $\times$. We call the elements of $\mathcal{A}_B^\times$ $\times$*-actions* and denote them generically by $\alpha_\times$ (e.g., a, $a \times b \in \mathcal{A}_B^\times$ but $a + b, a \times b + c, a \cdot b \notin \mathcal{A}_B^\times$ and $\mathbf{0}, \mathbf{1} \notin \mathcal{A}_B^\times$). Note that $\mathcal{A}_B^\times$ is finite because there is a finite number of basic actions in $\mathcal{A}_B$ which may be combined with the synchrony operator $\times$ in a finite number of ways (due to the weak idempotence of $\times$ over basic actions; see axiom (18) of Table 2). Note the inclusion of sorts $\mathcal{A}_B \subseteq \mathcal{A}_B^\times \subset \mathcal{A}$. 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: $+ < \cdot < \times < ^*$.

Axioms (14)-(17) give the properties of $\times$ to be associative, commutative, with identity element $\mathbf{1}$, and annihilator element $\mathbf{0}$ (i.e., $(\mathcal{A}, \times, \mathbf{1}, \mathbf{0})$ is a commutative monoid with an annihilator element). Axioms (14) and (15) basically say that the syntactic ordering of actions in a $\times$-action does not matter. Axiom (18) defines $\times$ to be weakly idempotent over the basic actions $a \in \mathcal{A}_B$. Note that this does *not* imply that we have an idempotent monoid. Axioms (19) and (20) define the distributivity of $\times$ over $+$. From axioms (14)-(20) together with (1)-(4) we conclude that $(\mathcal{A}, +, \times, \mathbf{0}, \mathbf{1})$ is a commutative and idempotent semiring (NB: idempotence comes from axiom (4), and the axiom (18) is just an extra property of the semiring).

At this point we give an informal intuition for the actions (elements) of $\mathcal{A}$: we consider that the actions are "done" by somebody (be that a person, a program, or an agent). One should not think exclusively of processes "executing" instructions as this is only one way of viewing the actions. Moreover, we do not discuss in this paper operational semantics nor bisimulation equivalences (like is done in SCCS).

With this non-algebraic intuition of actions we can elaborate on the purpose of $\times$, which models the fact that two actions are *done at the same time*. Doing actions at the same time should not depend on the syntactic ordering of the concurrent actions; thus the associativity and commutativity axioms (14) and (15) of $\times$. Intuitively, if a component does a skip action $\mathbf{1}$ then this should not be visible in the synchronous action of the whole system (thus the axiom (16)); whereas, if a component fails (i.e., does action $\mathbf{0}$) then the whole system fails (thus the axiom (17)).

Particular to $\times$ is the axiom (18) which defines a weak form of idempotence for the synchrony operator. The idempotence is natural for basic actions but it is not desirable for complex actions. Take as example a choice action performed synchronously with itself, $(a+b)\times(a+b)$. The first entity may choose a and the second entity may choose b thus performing the synchronous action $a\times b$. Therefore, the complex action is the same as $a+a\times b+b$ (by the distributivity axiom (19), the commutativity of $\times$ and $+$, idempotence of $\times$ over basic actions (18), and idempotence of $+$).

Particular to our concurrency model is axiom (21) which synchronizes sequences of actions by working in steps given by the $\cdot$ constructor. This encodes the synchrony model.

Note that there is no axiom relating the $\times$ with the Kleene star. There is no need as the relation is done by the synchrony axiom (21) and the fact that $\alpha^* = \mathbf{1} + \alpha\cdot\alpha^*$ from the axioms of *. Moreover, when combining two repetitive actions synchronously $\alpha^*\times\beta^*$ the synchrony operator will go inside the * operator. This results in an action inside * which has a maximum of $|\alpha|\times|\beta|$ steps (i.e., number of $\cdot$ applications). This is strongly related to the dimension of the automaton constructed in Theorem 2.21 to handle the $\times$ (the size of the new automaton is the size of the cartesian product of the two smaller automata; i.e., it is the product of the number of states).

Definition 2.2. *Consider $SKA \vdash \alpha = \beta$ to mean that the SKA equation can be deduced from the axioms of SKA using the standard rules of equational reasoning (reflexivity, symmetry, transitivity, and substitution), instantiation, and introduction and elimination of implication. Consider henceforth the relation $\equiv \subseteq T_{SKA}\times T_{SKA}$ defined as: $\alpha\equiv\beta \Leftrightarrow SKA\vdash\alpha=\beta$.*

Remark: The proof that $\equiv$ is a congruence is straightforward, based on the deduction rules, and we leave it to the reader.

Definition 2.3 (demanding relation). *We call $<_\times$ the demanding relation and define it as:*

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

In this paper we say that β is more demanding *than α iff $\alpha <_\times \beta$. We denote by $\leq_\times$ the relation $<_\times \cup =$ (i.e., $\alpha \leq_\times \beta$ iff either $\alpha <_\times \beta$ or $\alpha = \beta$).*

The intuition is that an action β is considered *more demanding* than another action α if we can see β as doing at the same time all the actions in α and something more. Consider the following examples: $\mathbf{1} <_\times a$, $a <_\times a \times b$, $a <_\times a$, $a \not<_\times b$, $a+b \not<_\times a+b$, $a+b \leq_\times a+b$, and $a \not<_\times b \times c$. We use $<_\times$ mainly to compare $\times$-actions of $\mathcal{A}_B^\times$ (similar to set inclusion). Note that the least demanding action is $\mathbf{1}$ (skipping means not doing any action). On the other hand, if we do not consider $\mathbf{1}$ then we have the basic actions of $\mathcal{A}_B$ as the minimal demanding actions; the basic actions are not related to each other by $<_\times$.

Proposition 2.4. *The relation $<_\times|_{\mathcal{A}_B^\times}$ is a partial order.*

Proof: For the relation $<_\times$ restricted to $\times$-actions the reflexivity is assured by the weak idempotence axiom (18) together with (14) and (15). The transitivity and antisymmetry are immediate and moreover, they hold for the whole set $\mathcal{A}$ of actions; e.g., for *transitivity* take any $\alpha, \beta, \gamma \in \mathcal{A}$ s.t. $\alpha <_\times \beta$ and $\beta <_\times \gamma$. Then it is the case that from $\alpha \times \beta = \beta$ and $\beta \times \gamma = \gamma$ we get $\alpha \times \gamma = \alpha \times \beta \times \gamma = \beta \times \gamma = \gamma$ which is the desired conclusion $\alpha <_\times \gamma$ (we used associativity of $\times$ and transitivity of the equality of actions). □

Corollary 2.5. *The relation $\leq_\times$ is a partial order for $\mathcal{A}$.*

Proof: Transitivity and antisymmetry were proven in Proposition 2.4 and reflexivity follows from the definition of $\leq_\times$. □

The conclusion of the two results above is that $<_\times$ is not a partial order for the whole set of actions (as opposed to the natural order $\leq$ of Kleene algebra). It is a partial order only when is restricted to $\times$-actions (the weak idempotence axiom (18) is used). On the other hand, when $<_\times$ is explicitly extended with equality we get a partial order $\leq_\times$ for the general actions.

Because $\times$ lacks idempotence for general actions we loose some monotonicity properties. In contrast to the natural order $\leq$, the operators $+$ and $\cdot$ are not monotone w.r.t. $<_\times$. The next result proves some weak monotonicity properties.

Proposition 2.6.

1. *If $\alpha_\times^i <_\times \beta_\times^i$ for all $1 \leq i \leq n$, then $\alpha_\times^1 \cdot \cdots \cdot \alpha_\times^n <_\times \beta_\times^1 \cdot \cdots \cdot \beta_\times^n \cdot \gamma$ where $\alpha_\times^i, \beta_\times^i \in \mathcal{A}_B^\times$ and $\gamma \in \mathcal{A}$.*
2. *If $\alpha_\times^i <_\times \beta_\times^j$ for all $i \leq n$ and $j \leq m$, then $(\alpha_\times^1 + \cdots + \alpha_\times^n) <_\times (\beta_\times^1 + \cdots + \beta_\times^m)$.*

Proof: For the first part of the proposition, the hypothesis is translated to $\alpha_\times^i \times \beta_\times^i = \beta_\times^i$ for all $1 \leq i \leq n$. We need to prove that $(\alpha_\times^1 \cdot \cdots \cdot \alpha_\times^n) \times (\beta_\times^1 \cdot \cdots \cdot \beta_\times^n \cdot \gamma) = \beta_\times^1 \cdot \cdots \cdot \beta_\times^n \cdot \gamma$. The synchrony axiom (21) can be applied on the left part of

the equality to combine the $\alpha_\times^i$ and $\beta_\times^i$ synchronously two by two which, by the hypothesis, become $\beta_\times^i$. We obtained exactly $\beta_\times^1 \cdot \dots \cdot \beta_\times^n \cdot \gamma$.

The proof of the second part of the proposition is similar. It makes use of the distributivity axiom (19) first, then uses the hypothesis in the same manner as before, and finally it contracts the same actions $\beta_\times^i$ with the axiom (4) of idempotence of $+$. □

Note that reflexivity for $<_\times$ is not a property of the general actions, but only of the $\times$-actions. Therefore, irreflexivity of $<_\times$ is not a property of the general actions either. On the other hand, we give some weaker results related to $<_\times$ applied to somehow more complex actions. Corollary 2.12 shows that for any *-free action α there exists a fixed point for the application of the $\times$ to the action itself. More precisely, define $\beta_0 = \alpha$ and $\beta_i = \beta_{i-1} \times \alpha$, then $\exists n \in \mathbb{N}$ and $\exists j < n$ s.t. $\beta_j = \beta_n$. This means that $\alpha <_\times \beta_n$ for any $n \geq j$. The proof uses the canonical representation of the action α. For example $(a+b)\times(a+b) = a+b+a\times b$ but on the other hand $(a + b + a \times b) \times (a + b) = a + b + a \times b$ (due to the weak idempotence of the $\times$ over $\mathcal{A}_B$, among other).

The canonical form (defined below) gives us a more structured way of viewing the *-free actions and makes easier the formulation and proof of the results from Corollary 2.12.

Definition 2.7 (canonical form for *-free actions). *We call* *-free actions *the terms of T_{SKA} constructed with the grammar below:*

$$\alpha \quad ::= \quad a \mid \mathbf{0} \mid \mathbf{1} \mid \alpha + \alpha \mid \alpha \cdot \alpha \mid \alpha \times \alpha$$

where $a \in \mathcal{A}_B$ is a basic action. Denote the set of all *-free actions by $\mathcal{A}^{\mathcal{D}}$.

We say that a *-free action α is in canonical form*, denoted by $\underline{\alpha}$, iff it has the following form:*

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

where $\alpha_\times^i \in \mathcal{A}_B^\times$ are pairwise distinct and $\underline{\alpha^i} \in \mathcal{A}^{\mathcal{D}}$ is in canonical form. The indexing set I is finite as the compound actions α are finite; i.e., there is a finite number of applications of the $+$ operator. Actions $\mathbf{0}$ and $\mathbf{1}$ are considered in canonical form.

Theorem 2.8. *For every *-free action $\alpha \in \mathcal{A}^{\mathcal{D}}$ there is corresponding $\underline{\alpha}$ in canonical form and equivalent to α (i.e., $SKA \vdash \alpha = \underline{\alpha}$).*

Proof: We use structural induction on the structure of the actions of $\mathcal{A}^{\mathcal{D}}$ 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 axioms of $\mathcal{CA}$. For convenience in the presentation of the proof, we define, for an action in canonical form $\underline{\alpha}$, the set $R = \{\alpha_\times^i \mid i \in I\}$ to contain all the $\times$-actions on the first "level" of $\underline{\alpha}$. Thus, we often use in the proof the alternative notation for the canonical form $\underline{\alpha} = +_{\alpha_\times^i \in R} \alpha_\times^i \cdot \underline{\alpha^i}$ which emphasizes the exact set of $\times$-actions on the first level of the action $\underline{\alpha}$. Often the R in the notation is omitted for brevity.

Base case:

a) When α is a basic action a of $\mathcal{A}_B$ it is immediately proven to be in canonical form just by looking at the definition of canonical form. Action a is in canonical form with the set $R = \{a\}$ and the $\cdot$ constructor is applied to a and to skip action $\mathbf{1}$ ($+_{a\in\{a\}}\, a \cdot \mathbf{1} \equiv a$).
b) The special actions $\mathbf{1}$ and $\mathbf{0}$ 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 follow from the associativity of the constructors.

Inductive step:

a) Consider $\alpha = \beta + \beta'$ is a compound action obtained by applying once the $+$ constructor. By the induction hypothesis β and β' are equivalent to canonical forms: $\beta \equiv +_{b_i\in B}\, b_i \cdot \beta_i$ and $\beta' \equiv +_{b'_j\in B'}\, b'_j \cdot \beta'_j$. Because of the associativity and commutativity of $+$, $\beta + \beta'$ is also in canonical form:

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

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 $\beta_a = \beta'_j$. Whenever $b_i = b'_j = a$ then $\beta_a = \beta_i + \beta'_j$. Because the inductive hypothesis states that all β_i and β'_j are in canonical form it follows that also β_a (which is either just a change of notation or a choice of two smaller canonical forms $\beta_i + \beta'_j$) are in canonical form.
b) Consider $\alpha = \beta \cdot \beta'$ with $\beta \equiv +_{b_i\in B}\, b_i \cdot \beta_i$ and $\beta' \equiv +_{b'_j\in B'}\, b'_j \cdot \beta'_j$ in canonical form. We now make use of the distributivity of $\cdot$ over $+$, and of the associativity of the $\cdot$ and $+$ constructors. In several steps α is transformed into a canonical form. In the first step α becomes

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

and, considering $|B| = m$, by the distributivity axiom (9), α becomes:

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

Subsequently $\cdot$ is distributed over all the members of the choice actions using axiom (8). In the end α becomes a choice of sequences, when we consider $|B| = m$ and $|B'| = k$:

$$\alpha \equiv b_1 \cdot \beta_1 \cdot b'_1 \cdot \beta'_1 \;+\; \ldots \;+\; b_m \cdot \beta_m \cdot b'_k \cdot \beta'_k.$$

This is clearly a canonical form because all actions $\beta_i \cdot b'_j \cdot \beta'_j$ are equivalent to canonical forms due to the inductive hypothesis. For the special case when $\beta_i = \mathbf{1}$ axiom (6) is applied to contract it to $b'_j \cdot \beta'_j$.

c) Consider $\alpha = \beta \times \beta'$ with $\beta \equiv +_{b_i \in B} b_i \cdot \beta_i$ and $\beta' \equiv +_{b'_j \in B'} b'_j \cdot \beta'_j$ in canonical form. First we use the distributivity axioms of $\times$ over $+$ and assume $|B| = m$ and $|B'| = k$, and from

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

we get

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

which distributes more to

$$\alpha \equiv (b_1 \cdot \beta_1) \times (b'_1 \cdot \beta'_1) \; + \; \ldots \; + \; (b_m \cdot \beta_m) \times (b'_k \cdot \beta'_k).$$

By applying the synchrony axiom (21) each summand is equivalent to

$$(b_i \cdot \beta_i) \times (b'_j \cdot \beta'_j) \equiv b_i \times b'_j \cdot \beta_i \times \beta'_j.$$

Clearly $b_i \times b'_j$ is a $\times$-action and the inductive hypothesis applied to the smaller β_i and β'_j yields that $\beta_i \times \beta'_j$ is equivalent to a canonical form. This means that each summand is equivalent to a canonical form, making α equivalent to a canonical form too.

□

Corollary 2.9. *For any *-free action α there exists an equivalent action $\beta \in \mathcal{A}^{\mathcal{D}}$ (i.e., $\beta \equiv \alpha$) which is of the following form:*

$$\underset{i \in I}{+} \underset{j \in J}{\cdot} \alpha_\times^{ij}.$$

Proof: The corollary basically says that any *-free action α is equivalent to a canonical form $\underline{\alpha}$ which is equivalent to an action which is a choice of sequences of synchronous actions $\alpha_\times$.

This is a simple consequence of the Theorem 2.8. Take the canonical form $\underline{\alpha} = +_{i \in I} \; \alpha_\times^i \cdot \underline{\alpha^i}$. A simple inductive argument suffices: the base case is trivial. For the inductive case consider

$$\underline{\alpha^i} = \underset{j \in J}{+} \underset{k \in K}{\cdot} \alpha_\times^{jk},$$

and just distribute the $\alpha_\times^i$ to obtain

$$\underset{j \in J}{+} (\alpha_\times^i \cdot \underset{k \in K}{\cdot} \alpha_\times^{jk}).$$

The associativity of $+$ finishes the proof. □

Lemma 2.10. *For any action that has the form of a sequence of synchronous actions, $\alpha = \alpha_\times^1 \cdot \cdots \cdot \alpha_\times^n$, then $\alpha \times \alpha = \alpha$. That is to say that $\times$ is idempotent and $<_\times$ is reflexive for actions of this form.*

Proof: The proof is trivial by using the synchrony axiom and the weak idempotence for $\times$. $\square$

Theorem 2.11. *For any *-free action α there exists $n \in \mathbb{N}$, which depends on α, for which $\alpha <_\times \alpha^{\times n}$, where $\alpha^{\times n}$ denotes the action obtained by putting n copies of α in synchronous combination.*

Proof: By Corollary 2.9 we consider $\alpha = +_{i\in I} \cdot_{j\in J} \alpha_\times^{ij}$.

Claim: $n = |I|$. We prove the claim using induction on $|I|$.

Base case: $|I| = 1$, means there is only one summand. Lemma 2.10 proves the case; i.e., because $\alpha = \cdot_{j\in J} \alpha_\times^{1j}$ then $\alpha \times \alpha = \alpha$ which is $\alpha <_\times \alpha$.

Inductive step: consider $\alpha = \alpha_k + \beta$ where $\alpha_k = \cdot_{k\in K} \alpha_\times^k$ is just a sequence of synchronous actions and $\beta = +_{i\in I} \cdot_{j\in J} \alpha_\times^{ij}$ with $|I| = n$ for which the induction hypothesis applies, and says that $\beta <_\times \beta^{\times n}$. The induction hypothesis translates to $\beta \times \beta^{\times n} = \beta^{\times n} = \beta^{\times(n+1)} = \beta^{\times(n+2)} = \ldots$.

We have to prove that $\alpha <_\times \alpha^{\times(n+1)}$. In other words, we need to prove that $\alpha^{\times(n+2)} = \alpha^{\times(n+1)}$. But $\alpha^{\times(n+1)} = \alpha \times \alpha^{\times(n)} = \alpha \times \alpha \times \alpha^{\times(n-1)} = (\alpha_k + \beta) \times (\alpha_k + \beta) \times \alpha^{\times(n-1)}$. Using distributivity and commutativity of $\times$, idempotence of $+$, and Lemma 2.10 for α_k we obtain $(\alpha_k + \alpha_k \times \beta + \beta^{\times 2}) \times (\alpha_k + \beta) \times \alpha^{\times(n-2)} = (\alpha_k + \alpha_k \times \beta + \alpha_k \times \beta^{\times 2} + \beta^{\times 3}) \times (\alpha_k + \beta) \times \alpha^{\times(n-3)}$. In the end we obtain $\alpha^{\times(n+1)} = \alpha_k + \alpha_k \times \beta + \alpha_k \times \beta^{\times 2} + \cdots + \beta^{\times(n+1)}$. Therefore, we have to prove that $\alpha_k + \alpha_k \times \beta + \alpha_k \times \beta^{\times 2} + \cdots + \alpha_k \times \beta^{\times(n+1)} + \beta^{\times(n+2)} = \alpha_k + \alpha_k \times \beta + \alpha_k \times \beta^{\times 2} + \cdots + \alpha_k \times \beta^{\times n} + \beta^{\times(n+1)}$. But this is easy using the induction hypothesis and the idempotence of $+$ to contract summands which are the same. $\square$

Corollary 2.12. *For any *-free action $\alpha \in \mathcal{A}^{\mathcal{D}}$ there exists $n \in \mathbb{N}$ for which for all $m \geq n$ it holds that $\alpha^{\times n} = \alpha^{\times m}$.*

Definition 2.13 (conflict and compatibility). *Consider a symmetric and irreflexive relation over the set of basic actions $\mathcal{A}_B$, which we call the* conflict relation *and denote by $\#_{\mathcal{C}} \subseteq \mathcal{A}_B \times \mathcal{A}_B$. The complement relation of $\#_{\mathcal{C}}$ is the symmetric and reflexive* compatibility relation *which we denote by $\sim_{\mathcal{C}}$ and define as:*

$$\sim_{\mathcal{C}} \triangleq \mathcal{U}_B \setminus \#_{\mathcal{C}}$$

where $\mathcal{U}_B = \mathcal{A}_B \times \mathcal{A}_B$ is the universal relation over basic actions.

When talking about actions *done at the same time* (i.e., synchronously) then it is natural to think about actions which *cannot* be done at the same time. For the basic actions this information is considered to be given a priori (by an oracle) and formalized as the conflict relation on $\mathcal{A}_B$. The intuition of the conflict relation is that if two actions are in conflict then the actions cannot be done synchronously. This intuition explains the need for the following equational implication:

$$(22)\quad a \#_{\mathcal{C}} b \to a \times b = \mathbf{0} \qquad \forall a, b \in \mathcal{A}_B.$$

The intuition of the compatibility relation is that if two actions are compatible then the actions can always be performed synchronously. Compatibility is obtained from the assumption that what is not in conflict is compatible. There is *no transitivity* of $\#_{\mathcal{C}}$ or $\sim_{\mathcal{C}}$; in general an action b may be in conflict with both a and c but still $a \sim_{\mathcal{C}} c$ (i.e., not necessarily $a \#_{\mathcal{C}} c$).

2.2. Standard interpretation over synchronous sets

We give the *standard interpretation* of the actions of $\mathcal{A}$ by defining a *homomorphism* $\hat{I}_{SKA}$ which takes any action of the *SKA* algebra into a corresponding *synchronous set* and preserves the structure of the actions given by the constructors.

Definition 2.14 (synchronous sets). *We consider a finite set denoted $\mathcal{A}_B$ (which for our discussion can be thought of as the basic actions). Consider a finite alphabet $\Sigma = \mathcal{P}(\mathcal{A}_B) \setminus \{\emptyset\}$ consisting of all* nonempty subsets of $\mathcal{A}_B$ *(denote them $x, y \in \Sigma$).* Synchronous strings *over Σ are elements of Σ^* including the empty string ϵ (denote them $u, v, w \in \Sigma^*$). A* synchronous set *is a subset of synchronous strings from Σ^* (denoted by A, B, C). Consider the following definitions and operations on synchronous sets:*

$$\begin{aligned}
\mathbf{0} &\triangleq \emptyset \\
\mathbf{1} &\triangleq \{\epsilon\} \\
A + B &\triangleq A \cup B \\
A \cdot B &\triangleq \{uv \mid u \in A, v \in B\} \\
A \times B &\triangleq \{u \times v \mid u \in A, v \in B\} \\
A^* &\triangleq \textstyle\bigcup_{n \geq 0} A^n
\end{aligned}$$

where uv is the concatenation of the two synchronous strings u and v, and $u \times v$ is defined as:

$$\begin{aligned}
&u \times \epsilon \triangleq u \triangleq \epsilon \times u \\
&u \times v \triangleq (x \cup y)(u' \times v') \text{ where } u = xu' \text{ and } v = yv',
\end{aligned}$$

and where $x, y \in \Sigma$ are sets of elements of $\mathcal{A}_B$. The powers A^n are defined as:

$$\begin{aligned}
A^0 &\triangleq \{\epsilon\} \\
A^n &\triangleq A \cdot A^{n-1}.
\end{aligned}$$

By convention when $A = \emptyset$ then $A^* = \{\epsilon\}$; thus A^* always contains the string ϵ. This operation is called the Kleene star.

Notation: Recall from formal languages the convention $u\epsilon = u = \epsilon u$. We abuse the notation and write a instead of the singleton set $\{a\}$ (also write $a \in \Sigma$). Moreover, we consider any subset of $\mathcal{A}_B$ as a synchronous string of length 1 and sometimes write a or x instead of u when the intention is clear from the context.

Theorem 2.15. *Any set of synchronous sets containing* **0** *and* **1** *and closed under the operations* $+, \cdot, \times, ^*$ *of Definition 2.14 is a synchronous Kleene algebra and forms a subalgebra of the powerset synchronous Kleene algebra of* Σ^*.

Proof: Routine check that the operations of Definition 2.14 obey the axioms of *SKA* from Table 2. Particular care needs to be taken for axioms (18) and (21) as they are defined on particular elements. Axiom (18) is defined only on the singleton sets $\{a\}$ with $a \in \mathcal{A}_B$ whereas axiom (21) is defined only on singleton synchronous sets $\{x\}$ with $x \in \Sigma$.

First we check that the full powerset of Σ^* is a *SKA*. It is easy to see that it is closed under $+$, $\cdot$, and *. It is also closed under $\times$ because if we take any two synchronous sets A and B then, by Definition 2.14, $A \times B$ is a set of strings where each element of a string is of the form $x \cup y \in \Sigma$ for $x, y \in \Sigma$; i.e., each element of $A \times B$ is a synchronous string.

We now prove that for the powerset algebra the operations of Definition 2.14 satisfy the *SKA* axioms of Table 2. The proofs for $+$, $\cdot$, and * are standard. In short, the $+$ operation over synchronous sets respects axioms (1)-(4) because it is defined in terms of the union operation $\cup$ over sets and **0** is defined as the empty set. The $\cdot$ operation is defined as in formal language theory and the proofs that it respects (5)-(9) are standard as we also have the convention $u\epsilon = u$ for the **1** case. For these cases and for the *, the fact that we work with synchronous strings makes no difference.

For the associativity axiom (14) of $\times$ we prove $u \in A \times (B \times C)$ iff $u \in (A \times B) \times C$, for any $A, B, C \subseteq \Sigma^*$. We prove the forward implication (the backward implication follows a similar reasoning). From the definition we have that $u = u_A \times (u_B \times u_C)$ with $u_A \in A, u_B \in B, u_C \in C$. We consider the general case in the definition of $\times$ over synchronous strings (where the particular case for ϵ follows from the proof of axiom (16)), and thus, $u_A = x_A u'_A$, $u_B = x_B u'_B$, and $u_C = x_C u'_C$. We have $x_A u'_A \times ((x_B \cup x_C)(u'_B \times u'_C)) \stackrel{Def.}{=} (x_A \cup (x_B \cup x_C))(u'_A \times (u'_B \times u'_C))$. Because $\cup$ for sets is associative and for the u' strings we follow an inductive argument we have that $u = ((x_A \cup x_B) \cup x_C)((u'_A \times u'_B) \times u'_C)$ which is the same as $(u_A \times u_B) \times u_C$; i.e., $u \in (A \times B) \times C$.

For the commutativity axiom (15) the proof is similar as above and it rests on the observation that the $\times$ operation of synchronous strings is commutative because it uses the set union $\cup$ at each element of the string.

For axioms (16) and (17) the proof comes from the definitions of **1** and **0** respectively. Consider $u \in A \times \{\epsilon\}$; then, by definition, $u = u_A \times \epsilon$ which, by the special case in the definition of $\times$ on synchronous strings, is equal to $u_A \in A$. For **0** it is clear from the definition of $\times$ on synchronous sets that $A \times \emptyset = \emptyset$.

The special axiom (18) applies only to singleton synchronous sets $\{a\}$ with $a \in \mathcal{A}_B$. It is easy to see that $\{a\} \times \{a\} = \{a\}$ because $a \cup a = a$ (i.e., $\cup$ over sets is idempotent).

The argument for the distributivity axioms (19) and (20) is similar to that for the $\cdot$ and $+$ operations. Consider $u \in A \times (B + C)$; then, by definition, $u = u_A \times u'$ with $u' \in B + C$. This means that either $u = u_A \times u_B$ or $u = u_A \times u_C$.

This is the same as $u \in (A \times B) + (A \times C)$.

For the synchrony axiom (21) the proof makes use of the fact that we need to consider only the special singleton synchronous sets of the form $\{x\}$ with $x \in \Sigma$. We need to prove $u \in (\{x\} \cdot A) \times (\{y\} \cdot B)$ iff $u \in (\{x\} \times \{y\}) \cdot (A \times B)$ for any two arbitrary singleton synchronous sets $\{x\}$ and $\{y\}$. By definition $u = xu_A \times yu_B = (x \cup y)(u_A \times u_B)$ for arbitrary $u_A \in A$ and $u_B \in B$. On the right hand side of the implication we have that $u = (x \cup y)(u_A \times u_B)$ for arbitrary $u_A \in A$ and $u_B \in B$. This completes the proof. □

After this proof it is clear that any subalgebra of the powerset algebra of Σ^* is an *SKA*. Most interesting is the smallest such algebra which contains $\mathbf{0}, \mathbf{1}$ and all $\{a\}$ for $a \in \mathcal{A}_B$, where $\mathcal{A}_B$ is some set which is finite and fixed beforehand. Denote this algebra by $\mathcal{ASS}$.

Definition 2.16 (standard interpretation). *An* interpretation *of SKA is a homomorphism with domain the term algebra T_{SKA}. We call* standard interpretation *the homomorphism $\hat{I}_{SKA} : T_{SKA} \to \mathcal{ASS}$. $\hat{I}_{SKA}$ is defined as the homomorphic extension of the map $I_{SKA} : \mathcal{A}_B \cup \{\mathbf{0}, \mathbf{1}\} \to \mathcal{ASS}$ to the whole set of actions T_{SKA}. I_{SKA} maps the generators of T_{SKA} into synchronous sets as follows:*

$$
\begin{aligned}
I_{SKA}(a) &= \{\{a\}\}, \forall a \in \mathcal{A}_B \\
I_{SKA}(\mathbf{0}) &= \emptyset \\
I_{SKA}(\mathbf{1}) &= \{\epsilon\}
\end{aligned}
$$

The homomorphic extension is standard:

$$
\begin{aligned}
\hat{I}_{SKA}(\alpha) &= I_{SKA}(\alpha), \forall \alpha \in \mathcal{A}_B \cup \{\mathbf{0}, \mathbf{1}\} \\
\hat{I}_{SKA}(\alpha + \beta) &= \hat{I}_{SKA}(\alpha) + \hat{I}_{SKA}(\beta) \\
\hat{I}_{SKA}(\alpha \cdot \beta) &= \hat{I}_{SKA}(\alpha) \cdot \hat{I}_{SKA}(\beta) \\
\hat{I}_{SKA}(\alpha \times \beta) &= \hat{I}_{SKA}(\alpha) \times \hat{I}_{SKA}(\beta) \\
\hat{I}_{SKA}(\alpha^*) &= \hat{I}_{SKA}(\alpha)^*
\end{aligned}
$$

The standard interpretation offers a method (i.e., a deterministic algorithm) for obtaining a model (as a set of synchronous strings) for an action of *SKA*. Just implement $\hat{I}_{SKA}$ as a recursive function on the structure of the actions stopping at the generators of T_{SKA}. From here the operations of $\mathcal{ASS}$ are applied upwards to generate the synchronous set corresponding to the initial action. Consider the example[2]: for $a, b, c \in \mathcal{A}_B$, $\hat{I}_{SKA}(a \times b \cdot c) = \hat{I}_{SKA}(a \times b) \cdot \hat{I}_{SKA}(c) = (\hat{I}_{SKA}(a) \times \hat{I}_{SKA}(b)) \cdot \hat{I}_{SKA}(c) = (\{a\} \times \{b\}) \cdot \{c\} \overset{Def.2.14}{=} \{\{a, b\}\{c\}\}$.

The standard models are the linking factor (semantically) between the syntactic elements of the algebra (i.e., between the equivalent actions). Moreover, the standard models are objects which are closer to our intuition, as sets of synchronous strings, and thus it is easier to work with (and compare) them.

[2]Recall the precedence of the operators $+ < \cdot < \times < ^*$.

Intuitively the skip action **1** means not performing any action and its interpretation as the set with only the empty string, which contains no basic action, goes well with the intuition. The fail action **0** is interpreted as the empty set following the intuition that there is no way of respecting a fail action.

Consider a $\times$-action $\alpha_\times = a_1 \times \ldots \times a_n$ with $a_i \in \mathcal{A}_B$ for all $1 \leq i \leq n$. The standard interpretation interprets $\alpha_\times$ as a singleton set $\hat{I}_{SKA}(\alpha_\times) = \{\{a_1, \ldots, a_n\}\}$ where the only string $w = \{a_1, \ldots, a_n\}$ has just one element of the alphabet Σ (i.e., one set of basic actions which form the $\times$-action $\alpha_\times$). Henceforth we denote sets like $\{a_1, \ldots, a_n\}$, coming from a $\times$-action $\alpha_\times$, by $\{\alpha_\times\}$. We use this notation, instead of just a general $x \in \Sigma$, when we want to make more explicit the set x. Moreover, we may apply set union to meant $\{\alpha_\times\} \cup \{\beta_\times\} = \{a_1, \ldots, a_n, b_1, \ldots, b_m\}$

2.3. Completeness and decidability

Regular expressions and finite automata (FA) are equivalent syntactic representations of regular languages (or regular sets as we call them) [33]. In this section we define finite automata that accept synchronous sets and prove an equivalent of Kleene's theorem. That is, for each action of T_{SKA} we can build a corresponding automaton which accepts the same synchronous set as the interpretation of the action. Using the translation as automata we give a combinatorial proof of completeness of the *SKA* w.r.t. the standard interpretation. Decidability follows from completeness and from the decidability of the inclusion problem for regular languages.

Definition 2.17 (automata on synchronous strings). Nondeterministic finite automata on synchronous strings *(NFA) are tuples* $A = (S, \Sigma, S_0, \rho, F)$ *consisting of a finite set of* states S*, the finite* alphabet of synchronous actions $\Sigma = \mathcal{P}(\mathcal{A}_B) \setminus \{\emptyset\}$ *(i.e., the powerset of the set of basic actions* $\mathcal{A}_B$ *minus the empty set), a set of* initial *designated states* $S_0 \subseteq S$*, a* transition function $\rho : \Sigma \to \mathcal{P}(S \times S)$*, and a set of final states* F*. An NFA is called* deterministic *(DFA) iff* $|S_0| = 1$ *and the transition function returns, for each label, not a relation over* S *but a partial function, i.e.,* $\rho : \Sigma \to (S \to S)$*.*

Notation: We denote the states of an automaton by either $s_i \in S$ or $i \in S$ with $i \in \mathbb{N}$. We often use s_0 to stand for the initial state and s_f to stand for a final state. Instead of a transition $(s_1, s_2) \in \rho(\{\alpha_\times\})$ we often use the graphical notation $s_1 \xrightarrow{\{\alpha_\times\}} s_2$ or the tuple notation $(s_1, \{\alpha_\times\}, s_2)$. A transition $s \xrightarrow{\epsilon} s'$ is called an ϵ-transition. The ϵ is not part of the alphabet Σ and thus it does not contribute to the accepted strings of an automaton. The label ϵ allows an automaton to take a transition without accepting any new symbol. This is useful later to give nice definitions of operations on automata. Further on we use set union over labels of the transitions; the special case for the ϵ should be understood as $\{\alpha_\times\} \cup \epsilon = \{\alpha_\times\}$. By the definition of ρ, NFA and DFA may not have ϵ-transitions (also called ϵ-free automata). In the sequel we need to talk about automata that do have ϵ-transitions and call them ϵ-NFA.

Definition 2.18 (acceptance). *A* run *of an automaton A is a finite sequence of transitions starting in the initial state, i.e., $s_0 \xrightarrow{\{\alpha_\times^1\}} s_1 \xrightarrow{\{\alpha_\times^2\}} s_2 \ldots \xrightarrow{\{\alpha_\times^n\}} s_n$. A run is called* accepting *if it ends in a final state, i.e., $s_n \in F$. An automaton* accepts a string w *iff there exists an accepting run s.t. $\{\alpha_\times^1\}\{\alpha_\times^2\}\ldots\{\alpha_\times^n\} = w$. The set of strings accepted by an automaton forms the* language *accepted by the automaton, denoted $\mathcal{L}(A)$ or just A.*

Our definition of NFA (and DFA) differs from the standard definition [33] in the choice of alphabet (we have sets of basic actions as labels) and the new definition of a synchrony product for these automata, which we see later. Because of this, many of the standard results for NFA and DFA which do not depend on the choice of alphabet can be adapted to our automata on synchronous strings.

Proposition 2.19 ([33]).

1. *DFAs and NFAs recognize the same class of languages; i.e., for any NFA there is a determinization procedure to generate a DFA which accepts the same language (and any DFA is also an NFA).*
2. *For any DFA one may use a Myhill-Nerode minimization procedure to obtain a minimal and unique DFA which accepts the same language as the initial automaton.*
3. *For any NFA we can construct an equivalent NFA which has only one final state and no transitions starting from the final state.*
4. *For any ϵ-NFA we can construct (using the standard ϵ-closure construction) an ϵ-free NFA which will accept the same language.*
5. *For any NFA with unique final state and no transitions starting from the final state we can remove all the ϵ-transitions which do not end in the final state (using a variation of ϵ-closure which does not consider the final state as part of the closures), and the resulting automaton will accept the same regular set.*

Because of the results of Proposition 2.19 we are free to work with NFA with one initial state, possibly one final state with no outgoing transitions, and which may have ϵ-transitions only ending in the final state; we denote the class of these automata by $A^{\mathcal{S}}$. Representants of this class are denoted $A^{\mathcal{S}}(\alpha)$ when they are related to α, as are the automata generated in Theorem 2.21 for some particular action α. This kind of automata facilitate the definition of the *synchrony product* operation on automata, corresponding to $\times$.

Corollary 2.20 (unique minimal automaton). *For any automaton on synchronous sets there exists a unique minimal deterministic automaton accepting the same synchronous set.*

Theorem 2.21 (actions to automata). *For any action $\alpha \in SKA$ we can construct an automaton $A^{\mathcal{S}}(\alpha)$ which accepts precisely $\hat{I}_{SKA}(\alpha)$.*

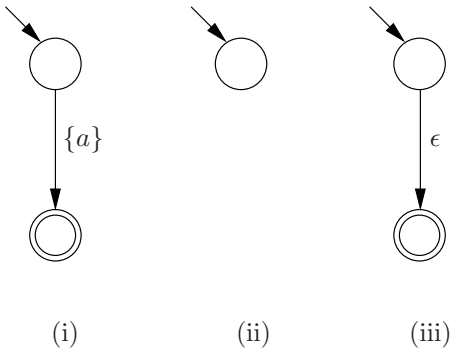

Figure 1: Automata corresponding to $a \in \mathcal{A}_B$, $\mathbf{0}$, and $\mathbf{1}$. Circles represent states, arrows represent transitions, pointed circles are the initial states, double circles are final states.

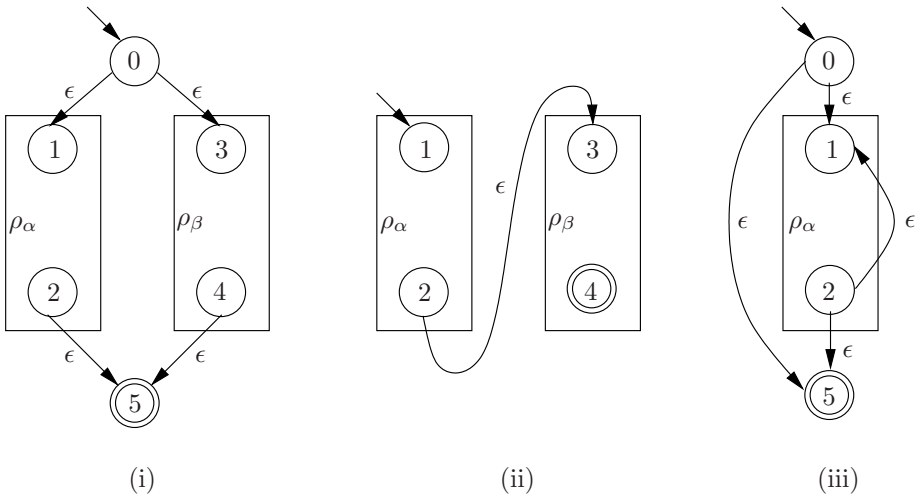

Figure 2: Automata corresponding to $\alpha + \beta$, $\alpha \cdot \beta$, and α^*.

Proof : The proof is adapted from [33] for the regular expressions operators ($+$, $\cdot$, and *) and we add a new construction for the synchrony operator $\times$. We use induction on the structure of actions.

Base case: For each action $a \in \mathcal{A}_B$, $\mathbf{0}$, and $\mathbf{1}$ we build the automata from Fig. 1 respectively (i), (ii), and (iii). It is easy to check that these automata are of $A^{\mathcal{S}}$ type and that they accept the corresponding synchronous sets.

Inductive step: For actions of the form $\alpha + \beta$, $\alpha \cdot \beta$, and α^* we use the standard constructions [33] pictured in Fig. 2 respectively (i), (ii), and (iii). These automata are constructed from the smaller automata corresponding to the smaller actions α and β and it is easy to check that they accept precisely the corresponding synchronous sets. Note that the automata of Fig. 2 are not

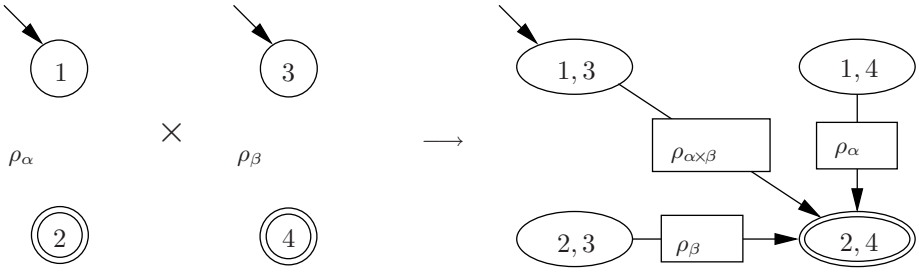

Figure 3: Example of automaton construction corresponding to $\alpha\times\beta$.

of $A^{\mathcal{S}}$ type, therefore, after each operation we need to apply Proposition 2.19(5) to remove the unwanted ϵ-transitions (e.g., in Fig. 2(i) states $0, 1, 3$ collapse into one, call it 013, and all transitions of the form $1 \xrightarrow{\{\alpha_\times\}} i$ or $3 \xrightarrow{\{\alpha_\times\}} i$ are replaced by a transition $013 \xrightarrow{\{\alpha_\times\}} i$, and the two ϵ-transitions are removed).

The new construction is the one for actions of the form $\alpha\times\beta$ which is schematically pictured in Fig. 3.[3] The automaton $A^{\mathcal{S}}(\alpha\times\beta)$ is constructed from the two smaller automata $A^{\mathcal{S}}(\alpha) = (S_\alpha, \mathcal{P}(\mathcal{A}^\alpha_B) \setminus \{\emptyset\}, s_1, \rho_\alpha, s_2)$ which accepts $\hat{I}_{SKA}(\alpha)$ and $A^{\mathcal{S}}(\beta) = (S_\beta, \mathcal{P}(\mathcal{A}^\beta_B) \setminus \{\emptyset\}, s_3, \rho_\beta, s_4)$ which accepts $\hat{I}_{SKA}(\beta)$, as follows:

$$A^{\mathcal{S}}(\alpha\times\beta) = (S_\alpha \times S_\beta, \mathcal{P}(\mathcal{A}^\alpha_B \cup \mathcal{A}^\beta_B) \setminus \{\emptyset\}, (s_1, s_3), \rho_{\alpha\beta}, (s_2, s_4)).$$

Note that states of $A^{\mathcal{S}}(\alpha\times\beta)$ are pairs of states of the old automata. Therefore, the initial state is the pair of the two initial states (s_1, s_3) and the final state is the pair of the old final states (s_2, s_4). The new transition relation is:

$$((s^\alpha_i, s^\beta_j), \gamma, (s^\alpha_k, s^\beta_l)) \in \rho_{\alpha\beta} \text{ iff either:}$$

- $\exists\gamma_1, \gamma_2$, s.t. $(s^\alpha_i, \gamma_1, s^\alpha_k) \in \rho_\alpha$ and $(s^\beta_j, \gamma_2, s^\beta_l) \in \rho_\beta$ and $\gamma_1 \cup \gamma_2 = \gamma$, or
- $(s^\alpha_i, \gamma, s^\alpha_k) \in \rho_\alpha$ and $s^\beta_j = s^\beta_l = s_4$, or
- $(s^\beta_j, \gamma, s^\beta_l) \in \rho_\beta$ and $s^\alpha_i = s^\alpha_k = s_2$.

The intuition behind this construction is that whenever both smaller automata can make a move then the new automaton can also make a move but labeled with the union of the two labels of the smaller automata. The last two cases are for when one of the states in the pair is a final state of one of the original automata. This is because the new automaton should be able to make a move whenever one of the smaller automata has stopped in a final state[4] and

[3] In Fig. 3 we picture only an example where there are no loop transitions for the initial states.

[4] $A^{\mathcal{S}}$ automata *stop* when reaching a final state because final states have no outgoing transitions.

the other automaton can still make a move. This behavior captures the application of synchrony to two synchronous strings of different lengths (the shorter one being accepted by the automaton that stops first). The new automaton $A^{\mathcal{S}}(\alpha \times \beta)$ has size $|S_\alpha| \times |S_\beta|$.

We need to prove now that the automaton $A^{\mathcal{S}}(\alpha \times \beta)$ accepts exactly the synchronous strings of the synchronous set $\hat{I}_{SKA}(\alpha \times \beta)$. We know that $\hat{I}_{SKA}$ is a homomorphism, thus $\hat{I}_{SKA}(\alpha \times \beta) = \hat{I}_{SKA}(\alpha) \times \hat{I}_{SKA}(\beta)$, and from the inductive hypothesis we know that $A^{\mathcal{S}}(\alpha)$ accepts exactly $\hat{I}_{SKA}(\alpha)$ and that $A^{\mathcal{S}}(\beta)$ accepts exactly $\hat{I}_{SKA}(\beta)$. Therefore we need to prove the following double implication:

$$w \in A^{\mathcal{S}}(\alpha \times \beta) \;\Leftrightarrow\; \exists u \in A^{\mathcal{S}}(\alpha) \text{ and } \exists v \in A^{\mathcal{S}}(\beta) \text{ s.t. } u \times v = w.$$

For the $\Leftarrow$ implication we assume that:

1. there exists an accepting run of $A^{\mathcal{S}}(\alpha)$ for u, i.e., $s_0^\alpha \xrightarrow{\{\alpha_\times^1\}} s_1^\alpha \xrightarrow{\{\alpha_\times^2\}} s_2^\alpha \ldots \xrightarrow{\{\alpha_\times^n\}} s_n^\alpha$ with $u = \{\alpha_\times^1\}\{\alpha_\times^2\}\ldots\{\alpha_\times^n\}$;
2. there exists an accepting run of $A^{\mathcal{S}}(\beta)$ for v, i.e., $s_0^\beta \xrightarrow{\{\beta_\times^1\}} s_1^\beta \xrightarrow{\{\beta_\times^2\}} s_2^\beta \ldots \xrightarrow{\{\beta_\times^m\}} s_m^\beta$ with $v = \{\beta_\times^1\}\{\beta_\times^2\}\ldots\{\beta_\times^m\}$;
3. $m \geq n$, and
4. $w = (\{\alpha_\times^1\} \cup \{\beta_\times^1\})(\{\alpha_\times^2\} \cup \{\beta_\times^2\}) \ldots (\{\alpha_\times^n\} \cup \{\beta_\times^n\})\{\beta_\times^{n+1}\} \ldots \{\beta_\times^m\}$.

From the construction of $A^{\mathcal{S}}(\alpha \times \beta)$ we need to find an accepting run (i.e., starting in (s_0^α, s_0^β) and ending in (s_n^α, s_m^β)) for the string w. It is easy to see that there are the following transitions in $\rho_{\alpha\beta}$: $(s_{i-1}^\alpha, s_{i-1}^\beta) \xrightarrow{\{\alpha_\times^i\} \cup \{\beta_\times^i\}} (s_i^\alpha, s_i^\beta)$, for $0 < i \leq n$, forming a run which starts in the initial state of $A^{\mathcal{S}}(\alpha \times \beta)$ and ends in (s_n^α, s_n^β) which accepts the first part of w. Because s_n^α is the final state of $A^{\mathcal{S}}(\alpha)$ then, from the construction of $\rho_{\alpha\beta}$, for each transition $s_j^\beta \xrightarrow{\{\beta_\times^{j+1}\}} s_{j+1}^\beta$, with $n \leq j < m$, in $A^{\mathcal{S}}(\beta)$ there is in $A^{\mathcal{S}}(\alpha \times \beta)$ the following transition $(s_n^\alpha, s_j^\beta) \xrightarrow{\{\beta_\times^{j+1}\}} (s_n^\alpha, s_{j+1}^\beta)$, with $n \leq j < m$. These transitions form a run in $A^{\mathcal{S}}(\alpha \times \beta)$ continuing the previous run and ending in the state (s_n^α, s_m^β) which is the final state of $A^{\mathcal{S}}(\alpha \times \beta)$, and it accepts the second part of w. Thus, we have found an accepting run of $A^{\mathcal{S}}(\alpha \times \beta)$ over the whole string w. Note that the assumption that $m \geq n$ is without loss of generality; if we were to take the opposite assumption then we would have worked in the automaton $A^{\mathcal{S}}(\alpha)$ for the second part of w and not in $A^{\mathcal{S}}(\beta)$ as we did by now.

For $\Rightarrow$ we assume that there exists an accepting run of $A^{\mathcal{S}}(\alpha \times \beta)$ for $w = \{w_\times^1\}\ldots\{w_\times^m\}$; let that be $(s_0^\alpha, s_0^\beta) \xrightarrow{\{w_\times^1\}} \ldots \xrightarrow{\{w_\times^m\}} (s_n^\alpha, s_m^\beta)$. By the construction of $A^{\mathcal{S}}(\alpha \times \beta)$ from the smaller automata $A^{\mathcal{S}}(\alpha)$ and $A^{\mathcal{S}}(\beta)$ we know that s_n^α is final state for $A^{\mathcal{S}}(\alpha)$ and s_m^β is final state for $A^{\mathcal{S}}(\beta)$. The choice of indices is wlog. We need to show that there are two accepting runs for $u \in A^{\mathcal{S}}(\alpha)$ and $v \in A^{\mathcal{S}}(\beta)$ such that $u \times v = w$.

Consider a first case when $s_0^\alpha = s_n^\alpha$ (wlog we work in $A^{\mathcal{S}}(\alpha)$). Because $A^{\mathcal{S}}$ automata have no outgoing transitions from the final state then it implies

that, in our run, $s_0^\alpha = s_i^\alpha$ for all $0 \leq i \leq n$ and $A^{\mathcal{S}}(\alpha)$ accepts only ϵ; thus $u = \epsilon$. Moreover, from the construction of $\rho_{\alpha\beta}$ we conclude that there exist the transitions $s_i^\beta \xrightarrow{\{w_\times^{i+1}\}} s_{i+1}^\beta \in \rho_\beta$ for all $0 \leq i < m$. Therefore we find in ρ_β an accepting run (ending in s_m^β) over $v = \{w_\times^1\} \ldots \{w_\times^m\}$. It is clear that $u \times v = \epsilon \times \{w_\times^1\} \ldots \{w_\times^m\} = w$.

Consider now the remaining case when $s_0^\alpha \neq s_n^\alpha$ and $s_0^\beta \neq s_m^\beta$ (i.e., neither s_0^α nor s_0^β are final states). Therefore, for the first transition $(s_0^\alpha, s_0^\beta) \xrightarrow{\{w_\times^1\}} (s_1^\alpha, s_1^\beta)$ we are in the first case of the construction of $\rho_{\alpha\beta}$ and thus, there exist the transitions $s_0^\alpha \xrightarrow{\{u_\times^1\}} s_1^\alpha \in \rho_\alpha$ and $s_0^\beta \xrightarrow{\{v_\times^1\}} s_1^\beta \in \rho_\beta$ s.t. $\{w_\times^1\} = \{u_\times^1\} \cup \{v_\times^1\}$. We continue with the subsequent transitions for w and find similar subsequent transitions for u and v until we reach a case like: $(s_{n-1}^\alpha, s_{n-1}^\beta) \xrightarrow{\{w_\times^n\}} (s_n^\alpha, s_n^\beta)$ where s_n^α is the final state of $A^{\mathcal{S}}(\alpha)$. (Note that we make, wlog, the assumption that we reach first the final state of $A^{\mathcal{S}}(\alpha)$; an analogous reasoning as the one below would work if we take the opposite assumption that we reach the final state of $A^{\mathcal{S}}(\beta)$ first.) In this case we have found the accepting run for $u = \{u_\times^1\} \ldots \{u_\times^n\}$. Moreover, if $s_n^\beta = s_m^\beta$ (i.e., we reach at the same time the final state of $A^{\mathcal{S}}(\beta)$) then we also found an accepting run for $v = \{v_\times^1\} \ldots \{v_\times^n\}$ and the proof is finished.

Consider now that $s_n^\beta \neq s_m^\beta$. Because s_n^α is final state there is no outgoing transition from it, and thus the only way to continue with the run for w is to have transitions of the form $(s_n^\alpha, s_j^\beta) \xrightarrow{\{w_\times^{j+1}\}} (s_n^\alpha, s_{j+1}^\beta)$ with $n \leq j < m$. Each of these transitions yields a transition for v in $A^{\mathcal{S}}(\beta)$: $s_j^\beta \xrightarrow{\{w_\times^{j+1}\}} s_{j+1}^\beta$. This process stops in the state (s_n^α, s_m^β) and therefore the run for v stops in the final state s_m^β of $A^{\mathcal{S}}(\beta)$. We have found the accepting run for $v = \{v_\times^1\} \ldots \{v_\times^n\}\{w_\times^{n+1}\} \ldots \{w_\times^m\}$. From the reasoning it is clear that $w = u \times v$. □

In [26] the completeness of Kleene algebra is proven by appealing to the representation of finite automata with matrices over arbitrary Kleene algebras. The most important construction is the * operation over matrices which basically gives the regular expressions encoding the regular languages accepted when going from each state of the automaton to every other state; a construction which comes from J.H. Conway [22]. In essence this construction is the algebraic equivalent of the combinatorial procedure of transforming an NFA into a regular expression [33]. In the algebraic approach to automata the regular language accepted by the automaton is obtained as (the interpretation of) a single regular expression.

The proof of completeness that we give follows similar ideas except that it uses a combinatorial argument. The motivation is that when giving semantics to deontic logic over synchronous actions or to the extension of PDL with synchrony we use the automata associated to actions, thus a combinatorial argument is more clarifying in this direction. The algebraic approach with matrices over synchronous Kleene algebras is based on definitions of operations on matrices

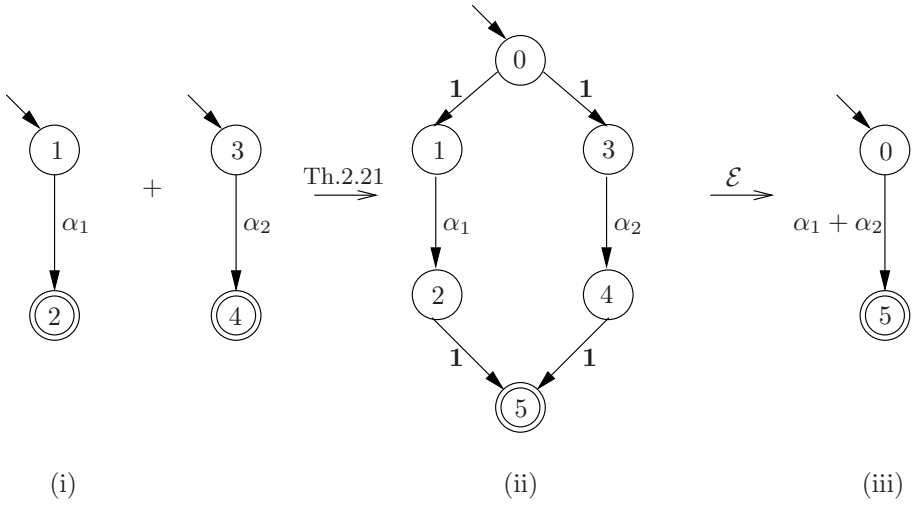

Figure 4: Application of $\mathcal{E}$ in Case 1 of Lemma 2.22.

corresponding to the operations on automata that we gave in Theorem 2.21.

We make use of the procedure of eliminating states, which generates a regular expression from an NFA [33]. Adapting the method of eliminating states to our automata on synchronous strings is trivial. Consider this method (which we denote $\mathcal{E}$) as a function which takes an automaton on synchronous strings $A^{\mathcal{S}}_{\alpha}$ and returns an action α of *SKA* s.t. $\hat{I}_{SKA}(\alpha) = \mathcal{L}(A^{\mathcal{S}}_{\alpha})$. Moreover, $\mathcal{E}$ considers the automata to have as labels actions from *SKA* instead of elements of the alphabet [33]. We consider the reader familiar with this standard technique for finite automata.

Lemma 2.22. *For all $\alpha \in T_{SKA}$ we have $\alpha \equiv \mathcal{E}(A^{\mathcal{S}}(\alpha))$.*

Proof: The proof of the lemma uses induction on the structure of the action. Moreover, we consider that $\mathcal{E}$ returns, instead of an action γ, an automaton with one initial and one final state, and one transition from the initial to the final state labeled by γ. (It is easy to see how $\mathcal{E}$ returns the corresponding γ from such an automaton.) This helps in the inductive reasoning below.

Base case: take the actions $\mathbf{0}$, $\mathbf{1}$, and $a \in \mathcal{A}_B$ and thus consider the $A^{\mathcal{S}}$ automata of Fig. 1. It is easy to see that the $\mathcal{E}$ procedure returns the regular expressions $\mathbf{0}$, $\mathbf{1}$, and $a \in \mathcal{A}_B$ respectively.

Inductive step:

Case 1: for $\alpha = \alpha_1 + \alpha_2$. The automaton $A^{\mathcal{S}}(\alpha_1 + \alpha_2)$ is obtained with the construction for + from Fig. 2(i) of Theorem 2.21 from the two automata $A^{\mathcal{S}}(\alpha_1)$ and $A^{\mathcal{S}}(\alpha_2)$. The inductive hypothesis says that $\alpha_1 \equiv \mathcal{E}(A^{\mathcal{S}}(\alpha_1))$; and we consider that $\mathcal{E}$ returns the automaton pictured in Fig. 4(i), and similar for α_2. It is easy to see how the automaton $A^{\mathcal{S}}(\alpha_1 + \alpha_2)$ is transformed into the one

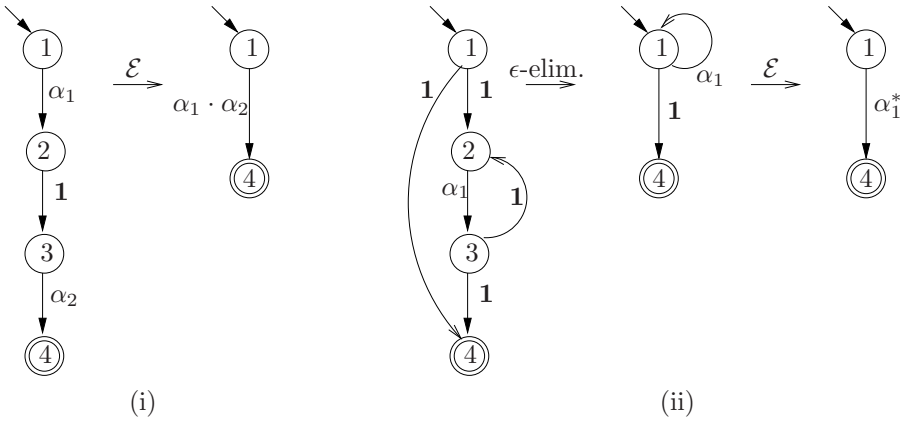

Figure 5: Application of $\mathcal{E}$ (i) for Case 2, (ii) for Case 3, of Lemma 2.22.

of Fig. 4(ii) by application of $\mathcal{E}$ to the smaller automata $A^{\mathcal{S}}(\alpha_1)$ and $A^{\mathcal{S}}(\alpha_2)$. From here $\mathcal{E}$ first eliminates states 1 and 2 to obtain a transition labeled with the action $\mathbf{1} \cdot \alpha_1 \cdot \mathbf{1} \equiv \alpha_1$ and then states 3 and 4 to obtain a transition labeled with $\mathbf{1} \cdot \alpha_2 \cdot \mathbf{1} \equiv \alpha_2$. Then it contracts the two resulting transitions into one labeled with $\alpha_1 + \alpha_2 \equiv \mathcal{E}(A^{\mathcal{S}}(\alpha_1 + \alpha_2))$ (as in Fig. 4(iii)).

Case 2: for $\alpha = \alpha_1 \cdot \alpha_2$. The automaton $A^{\mathcal{S}}(\alpha_1 \cdot \alpha_2)$ is obtained from the two automata $A^{\mathcal{S}}(\alpha_1)$ and $A^{\mathcal{S}}(\alpha_2)$ as in Theorem 2.21. Using the same inductive hypothesis as in Case 1 the procedure $\mathcal{E}$ is first applied to the two smaller automata and ends up with the automaton on the left of Fig. 5(i). By further eliminating the middle states we obtain $\mathcal{E}(A^{\mathcal{S}}(\alpha_1 \cdot \alpha_2)) = \alpha_1 \cdot \mathbf{1} \cdot \alpha_2 \equiv \alpha_1 \cdot \alpha_2$.

Case 3: for $\alpha = \alpha_1^*$. Take the automaton of Fig. 2(iii) which is constructed from $A^{\mathcal{S}}(\alpha_1)$ using the * operation from Theorem 2.21. By the same inductive hypothesis as in Case 1 we consider that $\mathcal{E}$ is first applied on $A^{\mathcal{S}}(\alpha_1)$ obtaining an automaton as in Fig. 5(ii), left. After this, the ϵ-elimination transforms it into a proper $A^{\mathcal{S}}$ from which $\mathcal{E}$ returns α_1^*. Note that $\mathcal{E}$ could have worked directly on the automaton with no ϵ-elimination and the result would have been the same (i.e., $\mathcal{E}(A^{\mathcal{S}}(\alpha_1^*)) = \mathbf{1} \cdot \alpha_1 \cdot (\mathbf{1} \cdot \alpha_1)^* \cdot \mathbf{1} + \mathbf{1} \equiv \alpha_1 \cdot \alpha_1^* + \mathbf{1} \equiv \alpha_1^*$).

Case 4: for $\alpha = \alpha_1 \times \alpha_2$. The proof for this case is more involved due to the local nature of the $\times$ operation. This local behavior can be seen both in the synchrony axiom (21) and in the definition of $\times$ over automata in Theorem 2.21 where the new transition relation is obtained by looking at each individual transition of the two smaller automata.

In short, the proof of this case uses again an inductive argument on the structure of the two actions α_1 and α_2. Essentially, we have to reason step by step on actions $\alpha_\times \in \mathcal{A}_B^\times$ because of the local behavior of $\times$, but we reason inductively.

Base case: $\alpha_1 = a$ and $\alpha_2 = b$. We again consider the automata $A^{\mathcal{S}}$ to have actions as labels, and instead of doing union of sets of basic actions we apply

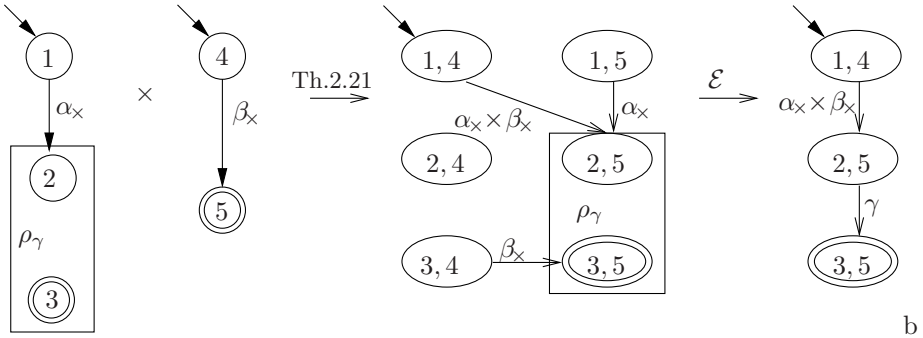

Figure 6: Schematic representation for Case 4.1 of Lemma 2.22.

$\times$ (i.e., instead of $\{a\} \cup \{b\}$ we now have $a \times b$). The automaton $A^{\mathcal{S}}(a \times b)$ is obtained from the two smaller automata $A^{\mathcal{S}}(a)$ and $A^{\mathcal{S}}(b)$ as in Fig. 3. It is easy to see that $\mathcal{E}(A^{\mathcal{S}}(a \times b))$ returns the action $a \times b$.

For the rest of the basis step consider α_1 or α_2 to be one of $\mathbf{0}$ or $\mathbf{1}$. The conclusion follows similarly as above.

From the base case is easy to see that we can reason with $\alpha_\times \in \mathcal{A}_B^\times$ actions as our basis, instead of just basic actions $a, b \in \mathcal{A}_B$. This is because the base case generates a $\times$-action, and if we apply it several times we get exactly all the $\times$-actions. In other words we can use an induction argument only using basic actions and the base case, and we prove the conclusion for $\times$-actions. Therefore, we may take the $\times$-actions as our basis. We do so in the rest of the proof.

Inductive step: We fix $\alpha_2 = \beta_\times$ with $\beta_\times \in \mathcal{A}_B^\times$, and take cases after α_1. This is the local behavior of our the proof; it works one step at a time (i.e., one transition at a time, labeled with $\times$-actions). When considering more complex actions for α_2 we still work with this assumption and treat the rest of the action inductively. Each time, the same cases as below need to be treated.

Case 4.1: for $\alpha_1 = \alpha_\times \cdot \gamma$. The automaton $A^{\mathcal{S}}(\alpha_1 \times \alpha_2)$ is obtained from the two smaller automata $A^{\mathcal{S}}(\alpha_\times \cdot \gamma)$ and $A^{\mathcal{S}}(\beta_\times)$ as in Fig. 6. The inductive hypothesis tells that $\mathcal{E}$ applied to the smaller automaton $A^{\mathcal{S}}(\gamma)$ returns one transition labeled with γ. Further applying $\mathcal{E}$ on the node $(2, 5)$ gives $\alpha_\times \times \beta_\times \cdot \gamma$. Note that the rest of the nodes do not play a role for $\mathcal{E}$ in generating the final action.

Case 4.2: for $\alpha_1 = \gamma_1 + \gamma_2$. For the $+$ operator we can reason about general actions because of the distributivity properties of the $\times$ on automata over $+$, in which case $\times$ simply considers the two sets of transitions from the two smaller automata separately. The argument is based on the case 4.1 before.

The automaton $A^{\mathcal{S}}(\alpha_1 \times \alpha_2)$ is built from the two smaller automata $A^{\mathcal{S}}(\gamma_1 + \gamma_2)$ and $A^{\mathcal{S}}(\beta_\times)$. The automaton $A^{\mathcal{S}}(\gamma_1 + \gamma_2)$ is constructed as a disjoint union of the two automata $A^{\mathcal{S}}(\gamma_1)$ and $A^{\mathcal{S}}(\gamma_2)$. Therefore, when combined synchronously with $A^{\mathcal{S}}(\beta_\times)$ the transition relation of $A^{\mathcal{S}}((\gamma_1 + \gamma_2) \times \beta_\times)$ is constructed inde-

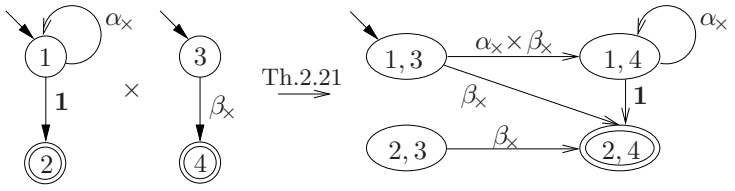

Figure 7: Schematic representation for Case 4.3.

pendently for the nodes of $A^{\mathcal{S}}(\gamma_1)$ and for the nodes of $A^{\mathcal{S}}(\gamma_2)$. Moreover, the nodes are separated into two disjoint parts. It is simple to see that this automaton comes from the disjoint union of $A^{\mathcal{S}}(\gamma_1 \times \beta_\times)$ and $A^{\mathcal{S}}(\gamma_2 \times \beta_\times)$. We now use Case 4.1 to apply $\mathcal{E}$ on each of these two disjoint automata to obtain transitions labeled respectively with $\gamma_1 \times \beta_\times$ and $\gamma_2 \times \beta_\times$. Then we finish by applying Case 1 and obtain $\mathcal{E}(A^{\mathcal{S}}((\gamma_1 + \gamma_2) \times \beta_\times)) = \gamma_1 \times \beta_\times + \gamma_2 \times \beta_\times \equiv (\gamma_1 + \gamma_2) \times \beta_\times$ by the distributivity axiom (20) of *SKA*.

Case 4.3: for $\alpha_1 = \alpha_\times^*$. For this case it is easy to look at the transition relation of the automaton $A^{\mathcal{S}}(\alpha_\times^* \times \beta_\times)$ and the inductive reasoning applies $\mathcal{E}$ on the smaller automata for $A^{\mathcal{S}}(\alpha_\times \times \beta_\times)$ and $A^{\mathcal{S}}(\alpha_\times)$ (as in Fig. 7). □

Theorem 2.23 (completeness of axiomatization). *For any two actions α and β it is the case that $SKA \vdash \alpha = \beta$ iff the corresponding synchronous sets $\hat{I}_{SKA}(\alpha)$ and $\hat{I}_{SKA}(\beta)$ are the same.*

Proof: The proof of the forward implication follows from the fact that $\mathcal{ASS}$ is a synchronous Kleene algebra (see Theorem 2.15). We use induction on the derivation and have as base case that the implication holds for the axioms of *SKA*, which was proven in Theorem 2.15. For the inductive step we consider the rules of equational reasoning, which are the same for both *SKA* and $\mathcal{ASS}$ (the details are omitted).

The proof of the converse implication is based on Lemma 2.22. Take two arbitrary actions $\alpha, \beta \in SKA$ s.t. $\hat{I}_{SKA}(\alpha)$ and $\hat{I}_{SKA}(\beta)$ denote the same synchronous set (i.e., $\hat{I}_{SKA}(\alpha) = \hat{I}_{SKA}(\beta)$). Construct, cf. Theorem 2.21, $A^{\mathcal{S}}(\alpha)$ and $A^{\mathcal{S}}(\beta)$ corresponding to the actions and accepting respectively $\hat{I}_{SKA}(\alpha)$ and $\hat{I}_{SKA}(\beta)$. Then transform them into unique deterministic automata, cf. Corollary 2.20. Because $\hat{I}_{SKA}(\alpha) = \hat{I}_{SKA}(\beta)$ we have that $A^{\mathcal{S}}(\alpha)$ and $A^{\mathcal{S}}(\beta)$ denote the same automaton (up to isomorphism of states). Now we apply $\mathcal{E}$ to obtain an action γ which is both $\gamma \equiv \alpha$ and $\gamma \equiv \beta$ (cf. Lemma 2.22). Therefore we have the conclusion $\alpha \equiv \beta$ (i.e., $SKA \vdash \alpha = \beta$). □

Theorem 2.24 (decidability). *The problem of deciding whether $\alpha = \beta$ in SKA is solved in quadratic time and is PSPACE-complete.*

Proof: The proof is a consequence of the completeness theorem. In order to test the equality of two actions we test the equality of the corresponding synchronous

sets $\hat{I}_{SKA}(\alpha)$ and $\hat{I}_{SKA}(\beta)$. This is done with the help of the translation of the actions into automata on synchronous sets. Then we use the method of [39] to get the PSPACE-completeness and a table-filling method to get a quadratic running time [33]. □

3. Synchronous Kleene Algebra with Boolean Tests

Tests add expressive power to Kleene algebra. Kleene algebra with tests is known to be more expressive than propositional Hoare logic [1] and it is the underlying algebraic formalism of the regular programs of PDL. On the other hand *KAT* is less expressive than PDL and different in time complexity too; *KAT* is PSPACE-complete whereas PDL is EXPTIME-complete. As algebras with the expressive power of PDL which are also extensions of Kleene algebras we mention dynamic algebras [23, 3] and modal Kleene algebras [40].

For this paper we are interested only in *KAT*. We follow the work of D. Kozen [13] and extend synchronous Kleene algebras with Boolean tests (denoted *SKAT*). This adds the expressive power of the Boolean tests to the synchronous actions of *SKA*.

SKAT inherits the expressivity of Kleene algebra with tests and it has the extra synchrony constructor. In the end of this section we show how we can reason about parallel programs with *SKAT* in the style of Owicki and Gries. Letting aside this particular application, *SKAT* is a general formalism which adds to the synchronous actions the power to make tests (of only Boolean expressivity). With *SKAT* one can express that at any point in a sequence of actions the system can stop and make a test (as a Boolean formula); if the test is successful then the execution can continue, otherwise it stops.

Definition 3.1. *Synchronous Kleene algebra with tests is given by* $SKAT = (\mathcal{A}, \mathcal{A}^?, +, \cdot, \times, ^*, \neg, \mathbf{0}, \mathbf{1})$ *which is an order-sorted algebraic structure with* $\mathcal{A}^? \subseteq \mathcal{A}$ *which combines the previously defined SKA with a Boolean algebra. The structures* $(\mathcal{A}^?, +, \cdot, \neg, \mathbf{0}, \mathbf{1})$ *and* $(\mathcal{A}^?, +, \times, \neg, \mathbf{0}, \mathbf{1})$ *are Boolean algebras and the Boolean negation operator* $\neg$ *is defined only on Boolean elements of* $\mathcal{A}^?$.

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

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

We do not go into details about the properties of a Boolean algebra as these are standard results. For a thorough understanding see [13] and references therein.

Note that the preference relation $\leq$ is defined over tests also and $\mathbf{1}$ is the most preferable test; i.e., $\forall \phi \in \mathcal{A}^?, \phi \leq \mathbf{1}$. It is natural to think of $\mathbf{1}$ as $\top$ because testing a tautology always succeeds; i.e., $\mathbf{1} \cdot \alpha = \alpha$, which says that the action α can always be performed after a $\mathbf{1}$ test. The $\mathbf{0}$ is seen as the dual $\bot$ meaning that testing a falsity never succeeds, and thus, any following action α is never performed; i.e., $\mathbf{0} \cdot \alpha = \mathbf{0}$ (the action sequence stops when it reaches the $\mathbf{0}$ test).

Definition 3.2 (extra axiom). *We give the equivalent of axiom (21) for tests:*

$$(21') \quad (\phi \cdot \alpha) \times (\varphi \cdot \beta) = (\phi \times \varphi) \cdot (\alpha \times \beta) \quad \forall \phi, \varphi \in \mathcal{A}^?.$$

Note that $\mathbf{1} \in \mathcal{A}^?$ and therefore this axiom allows sequences of actions with $\mathbf{1}$, which was not the case in axiom (21). On the other hand $\mathbf{1}$ is dealt with only in conjunction with another test, and not with another action. In this way the extra axiom (21′) still avoids interleaving; synchronous actions cannot be reduced to interleavings. Particular instances of this axiom are $\alpha \times \phi = \phi \cdot \alpha$ and $\phi \times \alpha = \phi \cdot \alpha$.

3.1. Interpretation over sets of guarded synchronous strings

Guarded strings have been introduced in [41] and have been used to give interpretation to Kleene algebra with tests [13]. Here we need an extension to *guarded synchronous strings* similar to the extension we gave in Section 2.2 from strings to synchronous strings. We intentionally overload several symbols as they have the same intuitive meaning but the particular definitions (adapted to guarded synchronous strings) are different.

Definition 3.3 (guarded synchronous strings). *Over the set of basic tests $\mathcal{A}^?_B$ we define* atoms *as functions $\nu : \mathcal{A}^?_B \to \{0,1\}$ assigning a Boolean value to each basic test. Consider the same finite alphabet Σ of all nonempty subsets of basic actions (denoted x, y as before). A* guarded synchronous string *(denoted by u, v, w) is a sequence*

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

where ν_i are atoms. We define $first(w) = \nu_0$ and $last(w) = \nu_n$.

Notation: Denote by $Atoms = \{0,1\}^{\mathcal{A}^?_B}$ the set of all atoms ν. We say that an atom *satisfies* a test ϕ (denoted $\nu \models \phi$) iff the truth assignment of the atom ν to the basic tests makes ϕ true. Note that for basic tests $\nu \models p$ iff $\nu(p) = 1$. We define two mappings over guarded synchronous strings: τ which returns the associated (unguarded) synchronous string; i.e., $\tau(w) = x_1 \dots x_n$ and π which returns the sequence of guards; i.e., $\pi(w) = \nu_0 \nu_1 \dots \nu_n$. Consider $Pref(\pi(w))$ to be the set of all prefixes of $\pi(w)$. Recall that when the $\times$-action $\alpha_\times$ is known or important then we use the notation $\{\alpha_\times\} \in \mathcal{P}(\mathcal{A}_B)$ instead of x.

Definition 3.4. *Consider* sets of guarded synchronous strings *denoted* A, B, C. *On these we define the following operations:*

$$\begin{aligned}
\mathbf{0} &\triangleq \emptyset \\
\mathbf{1} &\triangleq \mathit{Atoms} \\
A + B &\triangleq A \cup B \\
A \cdot B &\triangleq \{uv \mid u \in A, v \in B\} \\
A \times B &\triangleq \{u \times v \mid u \in A, v \in B\} \\
A^* &\triangleq \textstyle\bigcup_{n \geq 0} A^n \\
\neg A &\triangleq \mathit{Atoms} \setminus A, \quad \forall A \subseteq \mathit{Atoms}
\end{aligned}$$

where $u = \nu_0^u x_1 \nu_1^u \dots x_m \nu_m^u$ *and* $v = \nu_0^v y_1 \nu_1^v \dots y_n \nu_n^v$ *are guarded synchronous strings. The operation* $\cup$ *is just union over sets. The* fusion product uv *of two guarded synchronous strings is defined iff* $\mathit{last}(u) = \mathit{first}(v)$ *and is* $uv = \nu_0^u x_1 \dots x_m \nu_0^v y_1 \nu_1^v \dots y_n \nu_n^v$ *with* $\nu_m^u = \nu_0^v$. *The synchrony operation on guarded synchronous strings* $u \times v$ *is defined iff* $\pi(u) \in \mathit{Pref}(\pi(v))$ *when* $n \geq m$, *and is:*

$$u \times v \triangleq \nu_0^v (x_1 \cup y_1) \nu_1^v (x_2 \cup y_2) \nu_2^v \dots (x_m \cup y_m) \nu_m^v y_{m+1} \nu_{m+1}^v \dots y_n \nu_n^v;$$

whereas when $m \geq n$, *we require* $\pi(v) \in \mathit{Pref}(\pi(u))$, *and in the definition above literally and everywhere, interchange* u *and* v, m *and* n, *and* x *and* y.

Note that guarded synchronous strings can also be a single atom in which case π returns the atom itself. In such a case when both u and v are atoms then $u \times v$ is defined only if u and v are the same atom. The same holds for $\cdot$ over two atoms. Thus, when we consider two sets A and B of only atoms, then $A \cdot B$ and $A \times B$ become just intersection of sets.

Definition 3.5 (interpretation [42]). *The interpretation of the guarded actions is defined as the homomorphism* $\hat{I}_{SKAT}$ *from* T_{SKAT} *into* $\mathcal{AGSS}$. $\hat{I}_{SKAT}$ *is the homomorphic extension of the map* $I_{SKAT} : \mathcal{A}_B \cup \mathcal{A}_B^? \cup \{\mathbf{0}, \mathbf{1}\} \to \mathcal{AGSS}$ *which maps the generators of* T_{SKAT} *as follows:*

$$\begin{aligned}
I_{SKAT}(a) &= \{\nu\{a\}\nu' \mid \nu, \nu' \in \mathit{Atoms}\}, \forall a \in \mathcal{A}_B \\
I_{SKAT}(p) &= \{\nu \in \mathit{Atoms} \mid \nu(p) = 1\}, \forall p \in \mathcal{A}_B^? \\
I_{SKAT}(\mathbf{0}) &= \emptyset \\
I_{SKAT}(\mathbf{1}) &= \mathit{Atoms}
\end{aligned}$$

The homomorphic extension is standard:

$$\begin{aligned}
\hat{I}_{SKAT}(\alpha) &= I_{SKAT}(\alpha), \forall \alpha \in \mathcal{A}_B \cup \mathcal{A}_B^? \cup \{\mathbf{0}, \mathbf{1}\} \\
\hat{I}_{SKAT}(\alpha + \beta) &= \hat{I}_{SKAT}(\alpha) + \hat{I}_{SKAT}(\beta) \\
\hat{I}_{SKAT}(\alpha \cdot \beta) &= \hat{I}_{SKAT}(\alpha) \cdot \hat{I}_{SKAT}(\beta) \\
\hat{I}_{SKAT}(\alpha \times \beta) &= \hat{I}_{SKAT}(\alpha) \times \hat{I}_{SKAT}(\beta) \\
\hat{I}_{SKAT}(\alpha^*) &= \hat{I}_{SKAT}(\alpha)^* \\
\hat{I}_{SKAT}(\neg\phi) &= \neg \hat{I}_{SKAT}(\phi)
\end{aligned}$$

Theorem 3.6. *The smallest set containing $\emptyset$, Atoms, and the sets corresponding to $\mathcal{A}_B$ and $\mathcal{A}_B^?$ (i.e., those sets returned by the I_{SKAT} of Definition 3.5), and closed under the operations $+, \cdot, \times, ^*, \neg$ of Definition 3.4 is a synchronous Kleene algebra with tests. (Denote it $\mathcal{AGSS}$.)*

Proof : By routine check of the axioms of Table 2 together with the extra axiom for tests (21′) and the axioms for the two Boolean algebras of Definition 3.1. For all the axioms a thorough proof would consider a double implication: $\forall w : w \in A \Leftrightarrow w \in B$, where $A = B$ is an axiom. Here we only discuss the proof and do not go into details.

Because $\mathbf{0} = \emptyset$ and $+$ is defined with $\cup$ the axioms (1)-(4) are as in Theorem 2.15. For axiom (5) we know from Definition 3.4 that any $w \in A \cdot (B \cdot C)$ is $w = w_A w_B w_C$ with $last(w_B) = first(w_C)$ and $last(w_A) = first(w_B w_C)$ which is the same as $last(w_A) = first(w_B)$. The same conditions hold for the right part of the axiom. For (6) recall that $\mathbf{1} = Atoms$ and thus, each $w \in A$ is also part of $A \cdot \mathbf{1}$ (and also $\mathbf{1} \cdot A$) because $last(w) \in Atoms$ (respectively $first(w) \in Atoms$). Moreover, all other combinations of $w \in A$ with some $\nu \in Atoms$ with $last(w) \neq \nu$ are not included in $A \cdot \mathbf{1}$. Thus, the two sets $A \cdot \mathbf{1}$ and A have exactly the same guarded synchronous strings. For axiom (7) the cartesian product has no element because $\mathbf{0} = \emptyset$. For the distributivity axioms (8) and (9) a guarded synchronous string $w \in A \cdot (B + C)$ is of the form $w = w_A w_{BC}$ where w_{BC} is either in B or in C. If $w_{BC} \in B$ then $w \in A \cdot B$ and thus $w \in (A \cdot B) + (A \cdot C)$. For axioms (10)-(13) of the Kleene *, see the related proof of [13] as the definition of * is essentially the same.

We need to check the axioms (14)-(21) for $\times$. We first check commutativity (15). We assume that $w \in A \times B$ and thus $w = w_A \times w_B$ and assume wlog that $|w_A| = n \leq m = |w_B|$ and thus $\pi(w_A) \in Pref(\pi(w_B))$. Then w looks like:

$$w = \nu_0^B (x_1^A \cup y_1^B) \nu_1^B (x_2^A \cup y_2^B) \nu_2^B \ldots (x_n^A \cup y_n^B) \nu_n^B y_{n+1}^B \nu_{n+1}^B \ldots y_m^B \nu_m^B.$$

On the other hand, under the same assumption $n \leq m$ we can combine $w_B \times w_A$ to obtain:

$$w_B \times w_A = \nu_0^B (y_1^B \cup x_1^A) \nu_1^B (y_2^B \cup x_2^A) \nu_2^B \ldots (y_n^B \cup x_n^A) \nu_n^B y_{n+1}^B \nu_{n+1}^B \ldots y_m^B \nu_m^B.$$

Clearly, by the commutativity of $\cup$ for sets we have $w = w_B \times w_A \in B \times A$.

To check for associativity (14) take $w \in A \times (B \times C)$ to be $w = w_A \times w_{BC}$ where $w_{BC} = w_B \times w_C$. Because we proved commutativity we can now assume wlog that $|w_B| = m \leq |w_C| = n$ (we can reorder the terms using commutativity s.t. our assumption holds). Therefore, from the definition we have $\pi(w_B) \in Pref(\pi(w_C))$ and it remains to check three cases depending on the dimension of w_A:

1. assume $|w_A| = k \leq |w_B| = m$. Using the definition of $\times$ and the associativity of $\cup$ then w_{BC} looks like:

$$w_{BC} = \nu_0^C (y_1^B \cup z_1^C) \nu_1^C \ldots \nu_m^C z_{m+1}^C \ldots z_n^C \nu_n^C,$$

and because $k \leq m$ then $\pi(w_A) \in \mathit{Pref}(\pi(w_{BC}))$ and w becomes:

$$w = \nu_0^C(x_1^A \cup y_1^B \cup z_1^C)\nu_1^C \ldots \nu_k^C(y_{k+1}^B \cup z_{k+1}^C)\nu_{k+1}^C \ldots \nu_m^C z_{m+1}^C \ldots z_n^C \nu_n^C.$$

Under the same assumption $k \leq m \leq n$ we can combine first $w_A \times w_B = w_{AB} = \nu_0^B(x_1^A \cup y_1^B)\nu_1^B \ldots \nu_k^B z_{k+1}^B \ldots z_m^B \nu_m^B$ and because $\pi(w_{AB}) \in \mathit{Pref}(\pi(w_B)) \in \mathit{Pref}(\pi(w_C))$ we can combine $w_{AB} \times w_C$ to obtain the same guarded synchronous string w.

2. assume $|w_B| = m \leq |w_A| = k \leq |w_C| = n$. We can combine $w_B \times w_C$ to obtain the w_{BC} from case 1. Because $|w_A| \leq |w_C|$ then $\pi(w_A) \in \mathit{Pref}(\pi(w_C)) = \mathit{Pref}(\pi(w_{BC}))$. Therefore we may combine $w_A \times w_{BC}$ and obtain:

   $$w = \nu_0^C(x_1^A \cup y_1^B \cup z_1^C)\nu_1^C \ldots \nu_m^C(x_{m+1}^A \cup z_{m+1}^C)\nu_{m+1}^C \ldots \nu_k^C z_{k+1}^C \ldots z_n^C \nu_n^C.$$

   On the other hand, when combining first $w_A \times w_B$ we obtain:

   $$w_{AB} = \nu_0^A(x_1^A \cup y_1^B)\nu_1^A \ldots \nu_m^A x_{m+1}^A \ldots x_k^A \nu_k^A.$$

   Because $\pi(w_{AB}) = \pi(w_A) \in \mathit{Pref}(\pi(w_C))$ we may combine $w_{AB} \times w_C$ to obtain w as before.

3. assume $|w_B| = m \leq |w_C| = n \leq |w_A| = k$. We combine $w_B \times w_C$ to obtain the w_{BC} from case 1. Because $|w_C| \leq |w_A|$ then $\pi(w_{BC}) = \pi(w_C) \in \mathit{Pref}(\pi(w_A))$. Therefore we may combine $w_A \times w_{BC}$ and obtain:

   $$w = \nu_0^A(x_1^A \cup y_1^B \cup z_1^C)\nu_1^A \ldots \nu_m^A(x_{m+1}^A \cup z_{m+1}^C)\nu_{m+1}^A \ldots \nu_n^A x_{n+1}^A \ldots x_k^A \nu_k^A.$$

   As in the case 2. we may combine $w_A \times w_B$ to obtain w_{AB} as before. Because $\pi(w_C) \in \mathit{Pref}(\pi(w_A)) = \mathit{Pref}(\pi(w_{AB}))$ we may combine $w_{AB} \times w_C$ to obtain the same w as before.

Checking (16) and (17) is less laborious. For (17) use the same argument as for (7). For (16) is easy to see that for any atom $\nu \in \mathit{Atoms}$, $|\nu| \leq |w|$ for any guarded synchronous string w. Thus, a check $\pi(\nu) \in \mathit{Pref}(\pi(w))$ becomes just the check $\nu = \mathit{first}(w)$. In $\mathbf{1} \times A$, because $\mathbf{1} = \mathit{Atoms}$, for each $w \in A$ we always find a $\nu \in \mathit{Atoms}$ to match $\mathit{first}(w)$. Therefore, we always find the w in $\mathbf{1} \times A$ (and in $A \times \mathbf{1}$ when we check for $\mathit{last}(w)$).

For the weak idempotence axiom (18) we work only with words of the form $\nu\{a\}\nu'$. It is easy to see that this axiom is respected. The proof for the distributivity axiom (19) follows an analogous argument as for (8). Axiom (20) is just a consequence of (19) and (15).

We now prove that the synchrony axiom (21) is respected. We consider $w \in (A_\times \cdot A) \times (B_\times \cdot B)$ where the sets $A_\times$ and $B_\times$ are obtained only from sets that interpret $a \in \mathcal{A}_B$ using only the $\times$ operation. For example, take two sets $\{\nu\{a\}\nu' \mid \nu, \nu' \in \mathit{Atoms}\}$ and $\{\nu\{b\}\nu' \mid \nu, \nu' \in \mathit{Atoms}\}$, then their synchronous combination is $\{\nu\{a, b\}\nu' \mid \nu, \nu' \in \mathit{Atoms}\}$. Note that only the action changes, whereas the atoms remain all the possible ones from Atoms. Therefore, $A_\times = \{\nu\{\alpha_\times\}\nu' \mid \nu, \nu' \in \mathit{Atoms}\}$, for some set of basic actions $\{\alpha_\times\} \subseteq \mathcal{A}_B$.

We have that $w = w_{A_\times} w_A \times w_{B_\times} w_B$ where $w_{A_\times} = \nu_A\{\alpha_\times\}\nu'_A$ and $w_{B_\times} = \nu_B\{\beta_\times\}\nu'_B$ s.t. $\nu'_A = \mathit{first}(w_A)$ and $\nu'_B = \mathit{first}(w_B)$. Moreover, $\nu_A = \nu_B$ and

$\nu'_A = \nu'_B$, and wlog we assume $\pi(w_{A_\times} w_A) \in \mathit{Pref}(\pi(w_{B_\times} w_B))$ which entails $\pi(w_A) \in \mathit{Pref}(\pi(w_B))$. Thus, the combination $w_A \times w_B \in A \times B$ and has $\nu'_A = \nu'_B$ as the first atom. We can also make the synchronous composition $w_{A_\times} \times w_{B_\times} \in A_\times \times B_\times$ which is $\nu_A\{\alpha_\times \cup \beta_\times\}\nu'_A$. Because ν'_A is the last atom of $w_{A_\times} \times w_{B_\times}$ and the first atom of $w_A \times w_B$ the concatenation $(w_{A_\times} \times w_{B_\times})(w_A \times w_B) \in (A_\times \times B_\times) \cdot (A \times B)$ and is the same as w.

The proof of the extra axiom (21′) follows a similar argument as for (21).

Checking that $+$, $\cdot$, and $\neg$ over tests satisfy the laws of Boolean algebra is standard [42]. Moreover, from the results above it is clear that $\times$ over tests behaves like $\cdot$ (i.e., like Boolean conjunction). □

3.2. Automata on guarded synchronous strings

Automata on guarded strings have been introduced in [42] as an extension of finite automata with transitions labeled with a test or with a basic action. These automata accept regular languages of guarded strings. We define here automata on guarded synchronous strings with the help of the definition of automata on synchronous strings from the previous section. The presentation that we give in this section is an alternative to the presentation of automata on guarded strings from [42]. The main motivation is that it makes simpler some definitions for our automata on guarded synchronous strings, and the proofs that lead to completeness become easier to present.

First we identify a class of automata which accept sets of atoms. We then define what we call *two level automata*, which accept guarded synchronous strings. Then we need to define the particular operations of fusion product and synchrony product (corresponding to respectively $\cdot$ and $\times$ over sets of guarded synchronous strings) in order to prove the equivalent of Kleene's theorem. Using this, the proof of completeness requires a similar argument as in Theorem 2.23.

The next result is folklore and we omit its proof, which is found in [43].

Proposition 3.7 ([33]). *For two finite sets A and B one can construct a class of finite automata which accept all and only (encodings of) functions $f : A \to B$. The language accepted by such an automaton is a set of functions. Denote the set of all such automata by $\mathcal{M}$ and one automaton by $M \in \mathcal{M}$.*

Corollary 3.8. *Automata of Proposition 3.7 defined on the particular sets $\mathcal{A}^?_B$ and $\{0, 1\}$ (for respectively A and B) accept all and only the sets of atoms.*

Corollary 3.9. *The automata of Proposition 3.7 are closed under the well-known operations on finite automata union (denoted $\cup$) and intersection (denoted $\cap$); and correspond respectively to union of sets of functions and intersection of sets of functions. The automata of Proposition 3.7 are not closed under concatenation.*

Definition 3.10 (automata on guarded synchronous strings). *Let $A^{\mathcal{G}}$ be a* two level finite automaton *$A^{\mathcal{G}} = (S, \mathcal{P}(\mathcal{A}_B), S_0, \rho, F, \lceil\cdot\rceil)$, consisting at the first level of a finite automaton on synchronous strings, $(S, \mathcal{P}(\mathcal{A}_B), S_0, \rho, F)$ as*

in Definition 2.17, together with a map $\lceil\cdot\rceil : S \to \mathcal{M}$. *The mapping associates with each state of the first level an automaton* $M \in \mathcal{M}$ *as defined in Proposition 3.7 which accepts atoms. The automata in the states make the second (lower) level. Denote the language of atoms accepted by* $\lceil s\rceil$ *with* $\mathcal{L}(\lceil s\rceil)$.

Definition 3.11 (acceptance). *Take the definitions and notations for automata on synchronous strings from Section 2.3. We say that* a guarded synchronous string w is accepted by a two level automaton $A^{\mathcal{G}}$ *iff there exists an accepting run of the first level automaton which accepts* $\tau(w)$ *and for each state* s_i *of the run there exists an accepting run of the automaton* $\lceil s_i\rceil$ *which accepts the corresponding atom* ν_i *of* w.

It is easy to see that automata on guarded synchronous strings can be considered as ordinary finite automata. The two level definition that we give is useful in defining the *fusion product* and *synchrony product* operations over these automata.

Definition 3.12 (fusion product). *Define the fusion product automaton for two automata over guarded synchronous strings* $A_1^{\mathcal{G}} = (S_1, \mathcal{P}(\mathcal{A}_B), S_0^1, \rho_1, F_1, \lceil\cdot\rceil_1)$ *and* $A_2^{\mathcal{G}} = (S_2, \mathcal{P}(\mathcal{A}_B), S_0^2, \rho_2, F_2, \lceil\cdot\rceil_2)$ *as*

$$A_{12}^{\mathcal{G}} = (S, \mathcal{P}(\mathcal{A}_B), S_0, \rho, F, \lceil\cdot\rceil)$$

where:

- $S = (S_1 \setminus F_1) \cup (S_2 \setminus S_0^2) \cup S'$;
- $S' = F_1 \times S_0^2$ *and for* $s \in S'$ *denote* $s|_{F_1} \in F_1$ *the first component, and* $s|_{S_0^2}$ *the second component;*
- $S_0 = S_0^1$;
- $F = F_2$;
- $\rho = (\rho_1 \setminus \{(s_1, a, s_2) \in \rho_1 \mid s_2 \in F_1\}) \cup (\rho_2 \setminus \{(s_1, a, s_2) \in \rho_2 \mid s_1 \in S_0^2\})$
  $\cup\{(s_1, a, s) \mid s \in S' \text{ and } (s_1, a, s|_{F_1}) \in \rho_1\}$
  $\cup\{(s, a, s_1) \mid s \in S' \text{ and } (s|_{S_0^2}, a, s_1) \in \rho_2\}$;
- $\forall s \in S'$, $\lceil s\rceil = \lceil s|_{F_1}\rceil \cap \lceil s|_{S_0^2}\rceil$.

The first two conditions ensure that we combine the final states of the first automaton with all the initial states of the second automaton, so to get all possible concatenations. The next two conditions set the initial states to be the initial states of $A_1^{\mathcal{G}}$ and the final states to be the final states of $A_2^{\mathcal{G}}$. The condition on the transition relation keeps all the transitions from both automata and modifies accordingly those transitions that have to do with the old final states of $A_1^{\mathcal{G}}$ and the old initial states of $A_2^{\mathcal{G}}$. The last condition makes sure that we concatenate only guarded synchronous strings that have the same atoms *last* and $first$ (i.e., we keep in the new nodes of S only those atoms that correspond

to both the old final nodes of $A_1^{\mathcal{G}}$ and to the old initial nodes of $A_2^{\mathcal{G}}$). Note the use of $\cap$ to denote the operation of intersection of automata (which also results in intersection of their accepted languages).

Definition 3.13 (synchrony product). *Consider two disjoint automata over guarded synchronous strings denoted* $A_1^{\mathcal{G}} = (S_1, \mathcal{P}(\mathcal{A}_B), S_0^1, \rho_1, F_1, \lceil\cdot\rceil_1)$ *and* $A_2^{\mathcal{G}} = (S_2, \mathcal{P}(\mathcal{A}_B), S_0^2, \rho_2, F_2, \lceil\cdot\rceil_2)$. *Apply to the top level of these automata the synchrony product as defined in Theorem 2.21. For the lower level automata of the nodes do their intersection:* $\forall (s_1, s_2) \in S_1 \times S_2$, $\lceil (s_1, s_2) \rceil = \lceil s_1 \rceil \cap \lceil s_2 \rceil$.

Proposition 3.14. *Automata on guarded synchronous strings are closed under union, fusion product, and synchrony product.*

Proof: The union operation for automata on guarded synchronous strings is the same as given in Theorem 2.21 for automata on synchronous strings where any new first level nodes that are added have associated automata accepting any atom (i.e., accepting the set *Atoms*). For the fusion product it is clear that the top level automaton remains an automaton on synchronous strings and in each node the intersection of the automata for guards also gives an automaton for guards (because of closure under intersection; see Corollary 3.9). The same observation applies to the synchrony product. □

Theorem 3.15 (translation into automata). *For any* $\alpha \in T_{SKAT}$ *there is a corresponding automaton* $A^{\mathcal{G}}(\alpha)$ *which accepts exactly the set of guarded synchronous strings* $\hat{I}_{SKAT}(\alpha)$ *(i.e., the interpretation of* α*).*

Proof: We follow a similar recursive construction of the automaton based on the structure of the actions as we did in Theorem 2.21. We give constructions for the basic actions (here the recursion stops) and for each action constructor of *SKAT*. In each case we have to show that the automaton accepts the same set of guarded synchronous strings as the interpretation of the action. To show this we use an inductive argument.

From Proposition 3.7 we know we can construct an automaton for guards to recognize a given set of atoms. More precisely we can construct an automaton which recognizes the set of atoms that make only the basic test p true, or make only *some* or *none* of the basic tests true.

Base case: For a basic test p the automaton consists of only one top level state s which is both the initial and the final state. The second level automaton $\lceil s \rceil$ is such constructed to accept all and only the atoms which make the basic test p true. It is clear that this automaton accepts the set of guarded synchronous strings $\{\nu \in Atoms \mid \nu(p) = 1\}$ which corresponds to $\hat{I}_{SKAT}(p)$.

For the special actions $\mathbf{0}$ and $\mathbf{1}$ the construction is similar to that for tests. For $\mathbf{0}$ the automaton $\lceil s \rceil$ accepts the empty set thus the whole automaton accepts $\hat{I}_{SKAT}(\mathbf{0}) = \emptyset$. For $\mathbf{1}$ the automaton $\lceil s \rceil$ accepts all possible strings (i.e., a universal automaton) encoding all possible atoms; thus the automaton $A^{\mathcal{G}}(\mathbf{1})$ accepts $Atoms = \hat{I}_{SKAT}(\mathbf{1})$.

For a basic action $a \in \mathcal{A}_B$ we construct an automaton which at the top level is as the automaton in Fig. 2(i) and at the second level the automata $\lceil s_1 \rceil$ and $\lceil s_2 \rceil$ both accept *Atoms*; thus $A^{\mathcal{G}}(a)$ accepts $\{\nu\{a\}\nu' \mid \nu, \nu' \in Atoms\} = \hat{I}_{SKAT}(a)$.

Inductive step: Corresponding to the action constructors $\cdot$ and $\times$ we have respectively the constructions of fusion product and synchrony product on automata given in Definition 3.12 and Definition 3.13.

Consider $\alpha = \alpha_1 \cdot \alpha_2$. By the inductive hypothesis we have $\mathcal{L}(A^{\mathcal{G}}(\alpha_1)) = \hat{I}_{SKAT}(\alpha_1)$ and $\mathcal{L}(A^{\mathcal{G}}(\alpha_2)) = \hat{I}_{SKAT}(\alpha_2)$. From Definition 3.5 we know that $\hat{I}_{SKAT}(\alpha) = \hat{I}_{SKAT}(\alpha_1) \cdot \hat{I}_{SKAT}(\alpha_2)$ where $\cdot$ is as in Definition 3.4. The construction for fusion product of Definition 3.12 generates $A^{\mathcal{G}}(\alpha)$ s.t. it accepts $w = w_1 w_2$ where $w_1 \in A^{\mathcal{G}}(\alpha_1)$ and $w_2 \in A^{\mathcal{G}}(\alpha_2)$. By the inductive hypothesis we have that $w_1 \in \hat{I}_{SKAT}(\alpha_1)$ and $w_2 \in \hat{I}_{SKAT}(\alpha_2))$. Moreover, $last(w_1) = first(w_2)$ because of the last constraint of Definition 3.12 (in the generation of the automaton $A^{\mathcal{G}}(\alpha)$). Therefore, w is contained in $\hat{I}_{SKAT}(\alpha_1) \cdot \hat{I}_{SKAT}(\alpha_2)$ cf. Definition 3.4.

It remains to prove the opposite inclusion; i.e., that for any two $w_1 \in \hat{I}_{SKAT}(\alpha_1)$ and $w_2 \in \hat{I}_{SKAT}(\alpha_2)$ we have that if $w_1 w_2 \in \hat{I}_{SKAT}(\alpha_1 \cdot \alpha_2)$ then $w_1 w_2 \in \mathcal{L}(A^{\mathcal{G}}(\alpha_1 \cdot \alpha_2))$. From the same inductive hypothesis we know that $w_1 \in A^{\mathcal{G}}(\alpha_1)$ and $w_2 \in A^{\mathcal{G}}(\alpha_2)$. Because $w_1 w_2 \in \hat{I}_{SKAT}(\alpha_1 \cdot \alpha_2)$ then we know (cf. Definition 3.4) that $last(w_1) = first(w_2)$. According to Definition 3.12 the last condition is satisfied for w_1 and w_2 and thus the string $w_1 w_2$ is accepted by the fusion product of $A^{\mathcal{G}}(\alpha_1)$ and $A^{\mathcal{G}}(\alpha_2)$.

Consider $\alpha = \alpha_1 \times \alpha_2$. We treat first the inclusion $\mathcal{L}(A^{\mathcal{G}}(\alpha)) \subseteq \hat{I}_{SKAT}(\alpha)$; the opposite inclusion is simple and follows a similar reasoning as in the case before and Theorem 2.21. By the inductive hypothesis we have $\mathcal{L}(A^{\mathcal{G}}(\alpha_1)) = \hat{I}_{SKAT}(\alpha_1)$ and $\mathcal{L}(A^{\mathcal{G}}(\alpha_2)) = \hat{I}_{SKAT}(\alpha_2)$. From Definition 3.5 $\hat{I}_{SKAT}(\alpha_1 \times \alpha_2) = \hat{I}_{SKAT}(\alpha_1) \times \hat{I}_{SKAT}(\alpha_2)$ where $\times$ is the operation of Definition 3.4 over sets of guarded synchronous strings. $A^{\mathcal{G}}(\alpha_1 \times \alpha_2)$ is constructed as the synchrony product (i.e., Definition 3.13) of the two smaller automata $A^{\mathcal{G}}(\alpha_1)$ and $A^{\mathcal{G}}(\alpha_2)$.

Consider a guarded synchronous string w accepted by $A^{\mathcal{G}}(\alpha_1 \times \alpha_2)$. From Definition 3.11 we know that w is accepted if there exists an accepting run of the top level automaton of $A^{\mathcal{G}}(\alpha_1 \times \alpha_2)$ on the synchronous string $\tau(w)$, and for each state s_i of this accepting run the lower level automata accept the corresponding i^{th} element of $\pi(w)$. Because Definition 3.13, of synchrony product, uses the same construction for the top level automata as in the unguarded case then Theorem 2.21 assures that the synchronous string accepted by $A^{\mathcal{G}}(\alpha_1 \times \alpha_2)$ comes from the synchronous composition of two strings u and v accepted by the smaller automata $A^{\mathcal{G}}(\alpha_1)$ respectively $A^{\mathcal{G}}(\alpha_2)$. Moreover, the synchrony product construction makes the intersection of the automata in the states of the two $A^{\mathcal{G}}(\alpha_1)$ and $A^{\mathcal{G}}(\alpha_2)$ therefore, we know that $\pi(u) \in Pref(\pi(v))$ (or interchange u with v depending on the lengths). Because of this the requirements of Definition 3.4 for $\times$ over guarded synchronous strings are satisfied for u and v and thus $w = u \times v \in \hat{I}_{SKA}(\alpha_1) \times \hat{I}_{SKA}(\alpha_2)$.

For the action constructors $+$ and * we have the standard constructions from Fig. 2(i) respectively Fig. 2(iii) defined in Theorem 2.21 which are independent of

our special definition of the two level automata on guarded synchronous strings. Their proofs are standard as for finite automata and we skip them. □

3.3. Completeness and decidability

Theorem 3.16 (Completeness). *For any two actions α and β of T_{SKAT} we have that $SKAT \vdash \alpha = \beta$ iff the corresponding sets of guarded synchronous strings $\hat{I}_{SKAT}(\alpha)$ and $\hat{I}_{SKAT}(\beta)$ are the same.*

Proof : The forward implication (or soundness) comes as a consequence of Theorem 3.6 (because the sets of guaranteed synchronous strings form a synchronous Kleene algebra with tests). Consider, for example, one case for axiom (14) when $\alpha = \alpha_1 \times (\alpha_2 \times \alpha_3)$ and $\beta = (\alpha_1 \times \alpha_2) \times \alpha_3$. From Definition 3.5 $\hat{I}_{SKAT}(\alpha_1 \times (\alpha_2 \times \alpha_3)) = \hat{I}_{SKAT}(\alpha_1) \times (\hat{I}_{SKAT}(\alpha_2) \times \hat{I}_{SKAT}(\alpha_3))$ and $\hat{I}_{SKAT}((\alpha_1 \times \alpha_2) \times \alpha_3) = (\hat{I}_{SKAT}(\alpha_1) \times \hat{I}_{SKAT}(\alpha_2)) \times \hat{I}_{SKAT}(\alpha_3)$. The equality of the two sets comes from the associativity of $\times$ over sets of guarded synchronous strings which was proven in Theorem 3.6.

For the backward implication take arbitrary $\alpha, \beta \in T_{SKAT}$ with $\hat{I}_{SKAT}(\alpha) = \hat{I}_{SKAT}(\beta)$. For the two actions construct the corresponding $A^{\mathcal{G}}(\alpha)$ and $A^{\mathcal{G}}(\beta)$ as in Theorem 3.15 to accept $\hat{I}_{SKAT}(\alpha)$ respectively $\hat{I}_{SKAT}(\beta)$. In Theorem 2.23 it was easy to apply the Myhill-Nerode minimization procedure on the equivalent deterministic automata for synchronous strings. Unfortunately, in the case of guarded synchronous strings our two levels definition makes it impossible to adapt the subset construction method for determinization of automata on guarded synchronous strings of Definition 3.10. When making the set construction it is not possible to decide for a new state (as a set of some old states) what is the associated automaton (i.e., which set of atoms should it accept?).

On the other hand, as we remarked before, the two level definition is just a trick to get the right definitions of operations for automata on guarded synchronous strings and to make easy the proof of Kleene's theorem for constructing an equivalent automaton for a $\hat{I}_{SKAT}(\alpha)$. After having obtained such an automaton we do not need this definition anymore and we can see the automaton as a special finite automaton on a special alphabet $\Sigma = \mathcal{P}(\mathcal{A}_B) \cup Atoms$. More precisely, for each state with its deterministic finite automaton accepting a set of atoms we can split it into two states with one transition between them for each atom accepted by the automaton in the old state. For such a finite automaton the standard subset construction works and the Myhill-Nerode procedure is then applicable to obtain a unique automaton accepting $\hat{I}_{SKAT}(\alpha)$ (respectively $\hat{I}_{SKAT}(\beta)$). Because of the assumption of the theorem these two automata, $A^{\mathcal{G}}(\alpha)$ and $A^{\mathcal{G}}(\beta)$, denote the same automaton up to isomorphism of states.

All that remains to do is to show that an equivalent of Lemma 2.22 holds in this case; i.e., that a similar method of eliminating states $\mathcal{E}$ works for automata on guarded synchronous strings too. This is not hard to see as the automata that we work now with are the same as automata on synchronous strings except that between each two transitions labeled with a $\times$-action $\{\alpha_\times\}$ there are all those transitions labeled with atoms. But the $\mathcal{E}$ procedure is not influenced by these (it

just concatenates them, as tests, to the synchronous actions, thus obtaining the guarded synchronous actions) and thus we get $\alpha \equiv \mathcal{E}(A^{\mathcal{G}}(\alpha)) = \mathcal{E}(A^{\mathcal{G}}(\beta)) \equiv \beta$. □

3.4. SKAT, Hoare logic, and shared-variables concurrency

Propositional Hoare Logic (PHL) is the version of Hoare logic which does not involve the assignment axiom explicitly [1]. PHL reasons about programs at a more abstract level where the assignment axiom instances are just particular cases of atomic actions. Kleene algebra with tests (KAT) subsumes PHL and has the same complexity (PSPACE-complete for the Horn theory with premises of the form $\alpha = \mathbf{0}$). Moreover, KAT is complete for relational valid Horn formulas (i.e., all the rules of PHL are theorems of KAT), whereas PHL is incomplete. For instance, the following valid inference cannot be derived in PHL:

$$\frac{\{\psi\}\ \textbf{if}\ \phi\ \textbf{then}\ \alpha\ \textbf{else}\ \alpha\ \{\psi\}}{\{\psi\}\ \alpha\ \{\psi\}}$$

Extensions of Hoare logic exist which treat procedure calls, goto jumps, pointers, or aliasing. Similar extensions can be devised for Kleene algebra with tests, e.g., some form of higher-order functions [44], non-local flow of control [45], or local variables [46]. We do not know about nested procedure calls and returns, which are known to be a context free property (and not regular properties as is the style of KAT).

Regarding expressivity, KAT can encode the while programs (and more) which correspond to the notion of *tail recursion* (or *iteration*) from programming languages. It is known that tail recursion is strictly less expressive than (full) recursion. Recursion can be encoded with while programs and a stack. This corresponds to the context-free languages as opposed to the regular languages where KAT resides. The stack can be expressed in First-Order Dynamic Logic (which is undecidable in general), but KAT relates only to the weaker Propositional Dynamic Logic.

$SKAT$ includes KAT and thus all these expressiveness issues hold for our $SKAT$ too. Encoding partial correctness assertions (PCAs) into $SKAT$ is done as in [1, 19]. The PCA $\{\phi\}\alpha\{\psi\}$ intuitively says that if the program α starts in a state where ϕ holds (i.e., the precondition is true) then, whenever the program terminates,[5] the postcondition ψ will hold. There are two equivalent ways of encoding PCAs in $SKAT$: $\phi\alpha\neg\psi = \mathbf{0}$ (it is not possible that program α starts with precondition ϕ and terminates with postcondition $\neg\psi$) or $\phi\alpha = \phi\alpha\psi$ (testing the postcondition ψ after termination of α, started with precondition ϕ, is superfluous).

The definition of *interference freedom* of Owicki and Gries [14] says that an action a does not interfere with another action α iff it does not change the

[5] To talk about termination (total correctness) we need to use an extended version of the Hoare logic (not considered in this paper).

postcondition of α and when interleaved at any point in α it does not change the precondition of the remaining actions (to be executed) of α. In [14] the actions that need checking for interference freedom are only the statements that can change the state of the system (i.e., `await` and `assignment`). In our case these correspond to the basic actions.

Interference freedom in the synchrony model of *SKAT* is given by the $\sim_{\mathcal{C}}$ relation (defined in terms of $\#_{\mathcal{C}}$) on the basic actions of $\mathcal{A}_B$. The difference is that in our case $\#_{\mathcal{C}}$ is given by an "oracle" whereas in [14] it is given by rules based on the syntax (i.e., the assignment axiom schemata, which in practice reduces to the instances of the axiom for each particular assignment). In the case of *SKAT* we assume an oracle because the basic actions have no assumed structure (basic actions are abstract entities). If particularized to assignments then the oracle giving the $\#_{\mathcal{C}}$ relation becomes the assignment axiom schemata.

Extending the conflict and compatibility relations to the whole actions of *SKAT* is left for future work, but initial ideas are presented in Section 5.2. If two basic actions are not interference free then they cannot be executed synchronously: $a \#_{\mathcal{C}} b \rightarrow a \times b = \mathbf{0}$ (cf. axiom (22) of Section 2.1). This is extended to arbitrary actions $\alpha \#_{\mathcal{C}} \gamma \rightarrow \alpha \times \beta = \mathbf{0}$ (if α and γ are not interference free then their synchronous execution yields the impossible action). The following example needs this assumption that γ interferes with neither α nor β.

Example 3.1. *Consider two programs: a conditional* **if** ϕ **then** α **else** β *and an arbitrary* γ*. The conditional is written in SKAT as* $\phi\alpha + \neg\phi\beta$*. Now put these two programs to run in parallel (synchronously) and therefore write* $(\phi\alpha + \neg\phi\beta) \times \gamma$*. The following equality follows from the axioms of SKAT:* $(\phi\alpha)\times\gamma + (\neg\phi\beta)\times\gamma = (\phi\alpha)\times(\mathbf{1}\gamma) + (\neg\phi\beta)\times(\mathbf{1}\gamma)$ *which by axiom (21′) and rules of Boolean algebra becomes* $\phi(\alpha\times\gamma) + \neg\phi(\beta\times\gamma)$*. If we write this back into the while language we get:* **if** ϕ **then** $\alpha\times\gamma$ **else** $\beta\times\gamma$*.*

4. Synchronous Kleene Algebras vs. Other Concurrency Models

We present two models of concurrency based on partial orders and compare them with *SKA* for expressivity issues. *SKA* does not belong to the class of models of concurrency that are based on interleaving, but more to the class of models based on partial orders. It turns out that pomsets (and thus event structures [7, 47]) are strictly more expressive than *SKA*. On the other hand, Mazurkiewicz traces [6] and *SKA* are incomparable. Finally we discuss a recent model of concurrency, called concurrent Kleene algebra [48], which is close related to *SKA*.

4.1. Mazurkiewicz trace theory

Definition 4.1 (Mazurkiewicz traces). *Consider a symmetric and irreflexive binary relation* $I_{\mathcal{A}_B}$ *called the independence relation (i.e., not causal) on a set of basic actions, say* $\mathcal{A}_B$*. Define* $\equiv_{\mathcal{A}_B}$ *as the least congruence in the monoid of strings over* $\mathcal{A}_B$*, i.e.,* $(\mathcal{A}_B^*, \cdot, \mathbf{1})$ *s.t. if* $(a, b) \in I_{\mathcal{A}_B}$ *then* $ab \equiv_{\mathcal{A}_B} ba$*. For arbitrary strings we say that* $u \equiv_{\mathcal{A}_B} v$ *iff* $\exists w_1 \dots w_n$ *with* $u = w_1$ *and* $v = w_n$

and $\forall i, \exists w', w'', \exists a, b$ *s.t.* $(a, b) \in I_{\mathcal{A}_B}$ *and* $w_i = w'abw''$ *and* $w_{i+1} = w'baw''$. *One equivalence class generated by* $\equiv_{\mathcal{A}_B}$ *is called a (Mazurkiewicz)* trace *and is denoted by* $[w]_{\equiv_{\mathcal{A}_B}}$ *(the representative* w *is said to generate* $[w]_{\equiv_{\mathcal{A}_B}}$*).*

If $u \equiv_{\mathcal{A}_B} v$ then u is a permutation of v. A trace represents a run of a concurrent system. On the other hand a trace encodes several possible sequential runs which are considered equivalent due to the independence of some of the basic actions involved. From this point of view Mazurkiewicz traces talk about a special form of interleaving. The independence relation makes two basic actions *globally* independent; i.e., the basic actions are independent of each other no matter their position on the sequential runs.

In *SKA* a $\times$-action $\alpha_\times \in \mathcal{A}_B^\times$ is interpreted as the set of basic actions that compose it, e.g., $\{a, b, c\}$. Taking the same view (with sets of equivalent interleavings) we can say that $\alpha_\times$ encodes all the possible interleavings of these basic actions, e.g., $\{abc, acb, bac, bca, cab, cba\}$. Therefore, in the context of *SKA* the following definitions apply to the independence relation of Mazurkiewicz traces.

Definition 4.2. *Define the relation* $I_{\mathcal{A}_B}$ *as: for all* $a, b \in \mathcal{A}_B$*, if* $a \sim_{\mathcal{C}} b$ *(i.e.,* $a \times b \neq \mathbf{0}$ *cf. axiom (22)) then* $(a, b) \in I_{\mathcal{A}_B}$*. Extend this to* $\times$*-actions* $\alpha_\times$ *to say that if* $\alpha_\times \sim_{\mathcal{C}} \beta_\times$ *(i.e.,* $\alpha_\times \times \beta_\times \neq \mathbf{0}$*) then* $\forall a \in \{\alpha_\times\}, b \in \{\beta_\times\} : (a, b) \in I_{\mathcal{A}_B}$*.*

Proposition 4.3. *For a* $\times$*-action* $\alpha_\times = a_1 \times \ldots \times a_n$*,* $I_{\mathcal{A}_B}$ *restricted to the basic actions of* $\alpha_\times$ *is a total relation; i.e.,* $\forall\ 0 < i, j \leq n : (a_i, a_j) \in I_{\mathcal{A}_B}$*.*

Proof: The proof is easy and uses a *reductio ad absurdum* argument. Suppose that for some $0 < i \leq n$ and $0 < j \leq n$ it holds that $(a_i, a_j) \notin I_{\mathcal{A}_B}$. This means that $a_i \#_{\mathcal{C}} a_j$ (i.e., $a_i \not\sim_{\mathcal{C}} a_j$, for otherwise, by Definition 4.2, would imply $(a_i, a_j) \in I_{\mathcal{A}_B}$ contradicting our assumption). By axiom (22) it means that $a_i \times a_j = \mathbf{0}$ which implies that $\alpha_\times = \mathbf{0}$ which contradicts the statement of the proposition. □

This proposition shows a first difference between the concurrency modelled in *SKA* and the concurrency of Mazurkiewicz traces. In the latter the independence relation is not necessarily total (i.e., it may be defined as partial); this fact allows for some basic action to move back and forth along the sequence of actions depending on which actions it is independent of. Therefore *SKA* cannot capture the concurrent behavior of Mazurkiewicz traces.

For general actions of *SKA* generated using also the $\cdot$ operator the definitions above are not sufficient any more. Consider this simple example: $(a \times b) \cdot a$ in *SKA* has the following intended sequential runs: $\{aba, baa\}$; whereas in Mazurkiewicz traces, because $(a, b) \in I_{\mathcal{A}_B}$ we get the following sequential runs: $[aba]_{\equiv_{\mathcal{A}_B}} = \{aba, baa, aab\}$. This shows that Mazurkiewicz traces cannot capture the concurrent behavior intended in *SKA* because we need the independence relation to be *local* to each sequential step of a concurrent run.

In conclusion, for Mazurkiewicz traces the independence relation is *global* and *partial*. If we take the similar view in *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 $\times$-actions as forming a set.

4.2. Pomsets

Pomsets have long been advocated by Pratt [8] and many of the initial theoretical results were published as [49]. The theory of pomsets is among the first in concurrency theory to make a distinction between events (E) and actions (A). A pomset is a partially ordered set of events labeled (non-injectively) by actions. Pomsets extend the idea of strings, which are linearly ordered multisets, to partially ordered multisets. 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. Formally a *pomset* is the isomorphism class (w.r.t. the events) of the structure $(E, A, <, \mu)$ where $\mu : E \to A$ is the labeling function of the events by action names.

Two events which are incomparable by $<$ are permitted to occur concurrently. An important 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 an even more elaborated structure (in [49] operations over pomsets are defined by replacing events (with the same action name) by new pomsets). Moreover, the view of time does not matter as events may occupy time points or time intervals with no difference to the theory. There is also a large number of operations defined over pomsets (see [8]), more than in the other theories we have seen.

A pomset describes only one execution of the concurrent system. A set of pomsets is called a *process* and describes the whole set of concurrent behaviors of a system (or process). Pomsets are more expressive then our synchronous actions. We know that a synchronous action represents a set of synchronous strings (as we called them). Each synchronous string is a particular pomset; formally it is a pomset where the partial order respects the constraint:

$$\text{all } \textit{maximal independent sets} \text{ are disjoint,} \quad \begin{array}{l} \textit{uniquely labeled}, \text{ and} \\ \textit{completely ordered}, \end{array} \tag{23}$$

where an *independent set of events* is $X \subseteq E$ s.t. $(e_i \not< e_j) \wedge (e_j \not< e_i)$ for all $e_i, e_j \in X$. An independent set is *uniquely labeled* iff the labeling function is injective on X; i.e., $\mu|_X$ is injective. Two independent sets X_i, X_j are *completely ordered* iff whenever there exist $e_i \in X_i$ and $e_j \in X_j$ s.t. $e_i < e_j$ then $e_i < e_j$ for all $e_i \in X_i$ and $e_j \in X_j$. Call such a pomset a *synchronous pomset*.

Theorem 4.4. *Synchronous strings are completely characterized by synchronous pomsets.*

Proof : We need to prove two implications: for any w a synchronous string as in Definition 2.14 there is a synchronous pomset simulating it; and for any synchronous pomset there is a synchronous string.

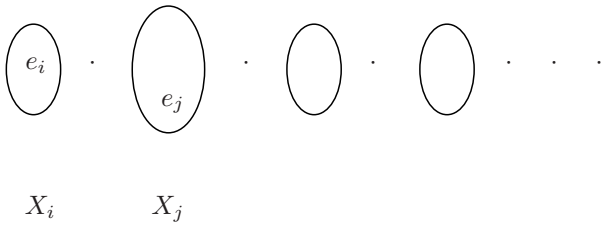

Figure 8: A synchronous string.

Consider a synchronous string as pictured in Fig. 8. It is formed of sets of incomparable events named by unique actions (because of the axiom (18)). These sets are pairwise disjoint and all the events in one set that follows after a $\cdot$ operator are in the relation $<$ with all the events that precede them (because of the associativity and non-commutativity of $\cdot$ we get the transitivity of the partial order). Therefore these are *independent sets* as in the definition above and are also *maximal*. The requirement of being *completely ordered* is clearly satisfied. Thus, we have the synchronous pomset.

For a synchronous pomset the fact that we consider the maximal independent sets to be disjoint gives much of the proof. These maximal independent sets make the $\times$-actions of the synchronous string. (In each $\times$-action all the basic actions composing it are considered independent.) The requirement of completely ordered ensures that one $\times$-action (i.e., all its composing basic actions) precedes the next $\times$-action (i.e., all the elements of the next independent set). Finally, the injective labeling ensures that $\times$-actions are actually interpreted as sets (and not as multisets), respecting axiom (18). $\square$

The good thing about the synchronous actions of *SKA* is that all the actions can be obtained (constructed) from a finite set of basic actions using a finite set of operations on actions. Is it the same situation for the synchronous pomsets?

Because of the $+$ operation the actions of *SKA* define *sets* of behaviors (of synchronous strings). Therefore we need to talk not of pomsets but of sets of pomsets (i.e., processes). The same $+$ operation exists for sets of pomsets (i.e., their union) which on synchronous pomsets behaves exactly like the $+$ in *SKA*. For the $\cdot$ there is the ; on pomsets. The extension of ; to sets of pomsets is exactly the same as the extension of $\cdot$ to sets of synchronous strings. Similarly we find the Kleene * for pomsets.

For the $\times$ of *SKA* we did not find a straightforward equivalent for synchronous pomsets. Moreover, we are not sure if there is a *pomset definable* operation (as in terminology of [49]). The first candidate was the concurrence operation $||$ but this breaks the *completely ordered* requirement. The *orthocurrence* operation on pomsets is also not good. We could not find a satisfactory new definition for $\times$ over pomsets because in order to enforce the conditions of synchronous pomsets we needed to look through the whole (infinite) structure of the partial

order on events starting with the smallest elements in the order.

4.3. Concurrent Kleene algebra

Recently Concurrent Kleene algebra (CKA) was proposed in [48] as a general formalism for reasoning about concurrent programs. CKA has, at first sight, striking resemblances with *SKA*. We discuss CKA in relation with *SKA*, focusing on the underlying ideas and intuitions of the two models.

CKA is defined as two quantales $(S, +, ;, 0, 1)$ and $(S, +, *, 0, 1)$ related by an exchange axiom ($;$ and $*$ correspond to respectively $\cdot$ and $\times$ in *SKA*). Quantales are idempotent semirings which are also complete lattices under the natural order $\leq$ of the semiring (i.e., have the extra constraint of a top element). What differentiates *SKA* from CKA is the synchrony axiom of the first and the exchange axiom of the second, and as we see later, also the choice of models.

Both algebras can model Hoare-style reasoning about sequential programs. Moreover, both algebras can reason about some form of concurrent programs: CKA can model Jone's rely/guarantee calculus, whereas *SKA* can reason about synchronous programs in the style of Qwicki and Gries (cf. Section 3.4).

The exchange axiom entails two properties of CKA relevant for our discussion:

$$(\alpha * \beta); (\alpha' * \beta') \leq (\alpha; \alpha') * (\beta; \beta') \tag{2}$$

$$\alpha; \beta \leq \alpha * \beta \tag{3}$$

Equation (2) is similar to the synchrony axiom. It is more general because it considers α and β general actions and not only $\times$-actions. On the other hand, it is less informative than the synchrony axiom because it only states inclusion of behaviors and not equality. One may read (2) as: "All behaviors coming from putting two concurrent compositions in sequence are captured by putting the respective sequences in concurrent composition."

Equation (3) states that the concurrent composition captures all the behavior of the sequential composition. This is the same as in the "concurrency as interleaving" approach where all the behaviors coming from all the possible interleavings are contained in the concurrent composition. CKA captures this because of (3) and the commutativity of $*$ (i.e., $\alpha; \beta + \beta; \alpha \leq \alpha * \beta$). Equation (3) does not hold in *SKA* and has no similar counterpart either. In *SKA* sequence composition and synchronous composition of two complex actions have different behaviors. *SKA* departs from the interleaving approach.

Looking at the models, we have seen the sets of synchronous strings of *SKA*, and how these are related to the partial orders models. For CKA the models are sets of traces, where a trace is just a set of events of E (i.e., models are just elements of $\mathcal{P}(\mathcal{P}(E))$). Moreover, E is equipped with a dependency relation $\rightarrow$ (no transitivity or acyclicity requirements as with partial orders).

In CKA the dependency relation is not manipulated, it is given. CKA processes specify subsets of events, and each subset has attached the predefined $\rightarrow$ restricted to its events only. In *SKA* and pomsets the partial order is changed

with each application of an operator; e.g., sequential composition adds dependencies. The approach of CKA is similar to that of separation logic where one reasons about a big (given) program by separating it into smaller independent programs. On the other hand, the partial orders model and *SKA* have a constructivist view where big programs are constructed from smaller programs (i.e. the partial order is constructed).

5. Conclusion

We have presented two algebraic structures for modelling synchronous actions. The first, synchronous Kleene algebra, is a combination between Kleene algebra and the synchrony model (i.e., we added the synchrony combinator $\times$). The second is the extension of synchronous Kleene algebras with Boolean tests. This gives more expressive power. We have seen the application of *SKAT* to reasoning about parallel programs with shared variables in the Hoare-style of Owicki and Gries, and we have hinted to the application of *-free actions in giving semantics to the $\mathcal{CL}$ contract logic.

We have focused on the theoretical aspects of the two new formalisms. Therefore, we have presented standard models (sets of respectively synchronous strings and guarded synchronous strings) and completeness results for the two algebras. The completeness proofs used a combinatorial argument based on two kinds of special automata that we defined. The equivalent of Kleene's theorem shows how we can obtain for each action a corresponding automaton which accepts the same set of models corresponding to the interpretation of the action.

5.1. Related Work

We now make some final discussions of formalisms which are somehow related to *SKA*, but we could not find strong arguments as we did in Section 4.

Shuffle is an operation over regular languages (basically over words) which preserves regularity. Shuffle has been used to model concurrency in [50, 49, 51] with a position between the interleaving approach and the partial orders approach. Shuffle is a generalization of interleaving similar to what we discussed for the Mazurkiewicz traces but it does not take into consideration any other relation on the actions/events that it interleaves. We can view $\times$ as some kind of *ordered shuffle*: the shuffling of two sequences of actions in *SKA* walks step by step (on the $\cdot$ operation) and shuffles the basic actions found (locally).

mCRL2 is a specification language for distributed systems built in the style of process algebras [52]. (mCRL2 is the successor of the μCRL language [53]) The semantics is given as SOS rules and a strong bisimulation relation is defined to capture the equality of processes. An axiomatization of the operators is given and (relative) completeness of the axiomatization w.r.t. the SOS semantics is shown. Recently a tool set has been released [54].

Many concepts of synchronous Kleene algebras are found in mCRL2. The building blocks of the language are a set of basic actions (parameterized by data types). The basic actions are grouped into sets of basic actions (called

multiactions) which are assumed to occur at the same time. The operation on multiactions is the same as $\times$ on $\mathcal{A}_B^{\times}$ in *SKA*. Over multiactions are defined the basic operators which are essentially the nondeterministic choice, sequence, and conditional (and a few nonessential for process references or for attaching time to a process in the form of a delay). There is no Kleene star concept but recursion is achieved, as in process algebras, through process definitions and process references. The rest of the operators are for parallel composition and synchronization (as in process algebra terminology), and additionally for restriction, blocking, renaming, and communication.

Analyzing an mCRL2 specification means linearization of the specification into what is called a linear process specification (which uses only the basic operators of mCRL2 in a restricted way). This linearization concept is the same as our models for the actions as sets of synchronous strings. The linearizations of mCRL2 are very close to our synchronous strings, except that they need to have more specific notions like the timers on processes (if any was specified in the original mCRL2 process) or the data arguments of the multiactions.

SKAT is a simple and clean formalism, but not as expressive as mCRL2. *SKAT* is tractable, and when used in the logical formalisms that we mentioned it still yields tractable logics. On the other hand one might need the addition of timing notions or of parameters (like the data types of mCRL2) to the actions, depending on the needs of the particular application domain.

5.2. Open problems

Continuations of the work presented here may take two directions: theory oriented and application oriented. From a theoretical point of view it would be interesting to see particular uses of the demanding relation $<_{\times}$ in the lines of thought that we drawn in the end of Section 2.1.

Details concerning the conflict relation $\#_{\mathcal{C}}$ were not given. An immediate question is how the conflict relation extends to the whole set of *SKA* actions? The first answer is to say that we add the equational implication (22) to the axioms of *SKA* (call this axiomatic system $SKA_{\#_{\mathcal{C}}}$). Two compound actions α and α' are said to be in the conflict relation $\alpha \#_{\mathcal{C}} \alpha'$ iff we cannot deduce $SKA \nvdash \alpha \times \alpha' = \mathbf{0}$ but we can deduce $SKA_{\#_{\mathcal{C}}} \vdash \alpha \times \alpha' = \mathbf{0}$. This solution relies on the decidability of $SKA_{\#_{\mathcal{C}}}$, which should not be difficult to establish since $\#_{\mathcal{C}}$ is a finite relation (defined on the finite set of basic actions $\mathcal{A}_B$). A related question is how does the theory (the results) change if we allow the set $\mathcal{A}_B$ to be possibly infinite?

Is there a canonical form for general actions of *SKA* (or *SKAT*) similar to what was done in Section 2 for the (restricted) *-free actions? Another interesting result, which is in the spirit of Kleene algebra theory, is to give a representation of automata on synchronous strings in terms of matrices over *SKA* and give an alternative proof of the completeness Theorem 2.23 similar to what is done in [26]. This involves the definition of an operation over matrices to simulate the synchrony product of finite automata over synchronous strings.

In this paper we have focused on the theoretical results of the two new formalisms *SKA* and *SKAT*. The application that we sketched are to logics based

on actions (deontic logic with synchronous actions and propositional Hoare logic with synchronous programs). We would like to see more investigations in this direction, with a more logical focus. The work on using the formalism of the *-free synchronous actions in the $\mathcal{CL}$ contract logic can be investigated more.

Related to this is a technically challenging problem: consider the equational system defining only the *-free actions in Definition 2.7 (i.e., axioms (1)-(9) of Table 1 together with axioms (14)-(21) of Table 2). Is there an algorithm to decide the *unification problem* for *-free synchronous actions? And what is its complexity? A more simple unification problem is to give an algorithm to find the substitution *solution* to the following:

$$\forall \alpha\beta \in \mathcal{CA},\ \alpha \times X = \beta, \text{ where } X \text{ is a } ^*\text{-free variable in } SKA.$$

Acknowledgements

I would like to thank Sergiu Bursuc for fruitful discussions and many comments on earlier drafts of this work. Thanks go also to Martin Steffen, Olaf Owe, and Gerardo Schneider for their useful comments. Several reviewers of earlier drafts of this paper have contributed with useful comments; special thanks go to the careful and meticulous JLAP reviewers.

References

[1] D. Kozen, On Hoare Logic and Kleene Algebra with Tests, Transactions on Computational Logic 1 (1) (2000) 60–76.

[2] V. R. Pratt, Process Logic, in: 6th Symposium on Principles of Programming Languages (POPL'79), ACM, 1979, pp. 93–100.

[3] V. R. Pratt, Dynamic Algebras as a Well-Behaved Fragment of Relation Algebras, in: C. H. Bergman, R. D. Maddux, D. L. Pigozzi (Eds.), Algebraic Logic and Universal Algebra in Computer Science, Vol. 425 of LNCS, Springer-Verlag, 1990, pp. 77–110.

[4] W. Kuich, A. Salomaa, Semirings, Automata, Languages, Springer-Verlag, Berlin, 1986.

[5] R. Milner, Calculi for Synchrony and Asynchrony, Theor. Comput. Sci. 25 (1983) 267–310.

[6] A. W. Mazurkiewicz, Basic Notions of Trace Theory, in: de Bakker et al. [55], pp. 285–363.

[7] M. Nielsen, G. D. Plotkin, G. Winskel, Petri Nets, Event Structures and Domains, in: G. Kahn (Ed.), Semantics of Concurrent Computation, Vol. 70 of LNCS, Springer, 1979, pp. 266–284.

[8] V. R. Pratt, Modeling Concurrency with Partial Orders, International Journal of Parallel Programming 15 (1) (1986) 33–71.

[9] R. de Simone, Higher-Level Synchronising Devices in Meije-SCCS, Theor. Comput. Sci. 37 (1985) 245–267.

[10] G. Berry, L. Cosserat, The ESTEREL Synchronous Programming Language and its Mathematical Semantics, in: S. D. Brookes, A. W. Roscoe, G. Winskel (Eds.), Seminar on Concurrency, Vol. 197 of LNCS, Springer, 1985, pp. 389–448.

[11] C. Prisacariu, G. Schneider, CL: An Action-based Logic for Reasoning about Contracts, in: M. Kanazawa, H. Ono, R. de Queiroz (Eds.), 16th Workshop on Logic, Language, Information and Computation (WOLLIC09), Vol. 5514 of LNCS, Springer, 2009, pp. 335–349.

[12] C. Prisacariu, G. Schneider, A Decidable Logic for Complex Contracts, Tech. rep., Dept. Informatics, Univ. Oslo, Norway (2009).

[13] D. Kozen, Kleene Algebra with Tests, ACM Transactions on Programming Languages and Systems (TOPLAS'97) 19 (3) (1997) 427–443.

[14] S. S. Owicki, D. Gries, An Axiomatic Proof Technique for Parallel Programs I, Acta Informatica 6 (1976) 319–340.

[15] G. H. Von Wright, An Essay in Deontic Logic and the General Theory of Action, North Holland Publishing Co., Amsterdam, 1968.

[16] K. Segerberg, A Deontic Logic of Action, Studia Logica 41 (2) (1982) 269–282.

[17] K. Aboul-Hosn, D. Kozen, KAT-ML: an Interactive Theorem Prover for Kleene Algebra with Tests, Journal of Applied Non-Classical Logics 16 (1-2) (2006) 9–34.

[18] C. A. R. Hoare, Fine-Grain Concurrency, in: A. A. McEwan, S. Schneider, W. Ifill, P. Welch (Eds.), Communicating Process Architectures 2007, Vol. 65 of Concurrent Systems Engineering Series, IOS Press, 2007, pp. 1–19.

[19] D. Harel, D. Kozen, J. Tiuryn, Dynamic Logic, MIT Press, 2000.

[20] D. Peleg, Concurrent Dynamic Logic (Extended Abstract), in: 7th ACM Symposium on Theory of Computing (STOC'85), ACM, 1985, pp. 232–239.

[21] S. C. Kleene, Representation of Events in Nerve Nets and Finite Automata, Automata Studies (1956) 3–41.

[22] J. H. Conway, Regular Algebra and Finite Machines, Chapman and Hall, 1971.

[23] D. Kozen, On the Duality of Dynamic Algebras and Kripke Models, in: Logic of Programs, Vol. 125 of LNCS, Springer-Verlag, 1979, pp. 1–11.

[24] D. Kozen, On Kleene Algebras and Closed Semirings, in: B. Rovan (Ed.), Mathematical Foundations of Computer Science (MFCS'90), Vol. 452 of LNCS, Springer, 1990, pp. 26–47.

[25] A. Salomaa, Two Complete Axiom Systems for the Algebra of Regular Events, Journal of ACM 13 (1) (1966) 158–169.

[26] D. Kozen, A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events, Information and Computation 110 (2) (1994) 366–390.

[27] E. Cohen, D. Kozen, F. Smith, The Complexity of Kleene Algebra with Tests, Tech. Rep. 1598, Cornell University (1996).

[28] E. Cohen, Using Kleene Algebra to Reason About Concurrency Control, Tech. rep., Telcordia (1994).

[29] D. Kozen, M.-C. Patron, Certification of Compiler Optimizations Using Kleene Algebra with Tests, in: J. W. Lloyd, V. Dahl, U. Furbach, M. Kerber, K.-K. Lau, C. Palamidessi, L. M. Pereira, Y. Sagiv, P. J. Stuckey (Eds.), Computational Logic (CL'00), Vol. 1861 of LNCS, Springer, 2000, pp. 568–582.

[30] D. Kozen, Kleene Algebra with Tests and the Static Analysis of Programs, Tech. Rep. 1915, Cornell University (2003).

[31] B. Möller, Calculating with Pointer Structures, in: R. S. Bird, L. G. L. T. Meertens (Eds.), Algorithmic Languages and Calculi, Vol. 95 of IFIP Conference Proceedings, Chapman & Hall, 1997, pp. 24–48.

[32] D. Kozen, Typed Kleene Algebra, Tech. Rep. 1669, CS Department, Cornell University (1998).

[33] J. E. Hopcroft, R. Motwani, J. D. Ullman, Introduction to Automata Theory, Languages, and Computation, 2nd Edition, Addison-Wesley, 2000.

[34] M. J. Fischer, R. E. Ladner, Propositional Modal Logic of Programs, in: 9th ACM Symposium on Theory of Computing (STOC'77), ACM, 1977, pp. 286–294.

[35] D. Kozen, The Design and Analysis of Algorithms, Springer-Verlag, 1997.

[36] G. Berry, G. Gonthier, The Esterel Synchronous Programming Language: Design, Semantics, Implementation, Science of Computer Programming 19 (2) (1992) 87–152.

[37] R. Milner, Communication and Concurrency, Prentice Hall, 1995.

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

[39] L. J. Stockmeyer, A. R. Meyer, Word Problems Requiring Exponential Time, in: 5th Annual ACM Symposium on Theory of Computing (STOC'73), ACM, 1973, pp. 1–9.

[40] J. Desharnais, M. Bernhard, G. Struth, Modal Kleene Algebra and Applications – A Survey, Journal of Relational Methods in Computer Science 1 (2004) 93–131.

[41] D. M. Kaplan, Regular Expressions and the Equivalence of Programs, J. Comput. Syst. Sci. 3 (4) (1969) 361–386.

[42] D. Kozen, Automata on Guarded Strings and Applications, in: J. T. Baldwin, R. J. G. B. de Queiroz, E. H. Haeusler (Eds.), Workshop on Logic, Language, Informations and Computation (WoLLIC'01), Vol. 24 of Matematica Contemporanea, Sociedade Brasileira de Matemática, 2003, pp. 117–139.

[43] C. Prisacariu, Extending Kleene Algebra with Synchrony, Tech. Rep. 376, Dept. Informatics, Univ. Oslo (2008).

[44] K. Aboul-Hosn, D. Kozen, Relational Semantics for Higher-Order Programs, in: T. Uustalu (Ed.), 8th International Conference on Mathematics of Program Construction (MPC'06), Vol. 4014 of LNCS, Springer, 2006, pp. 29–48.

[45] D. Kozen, Nonlocal Flow of Control and Kleene Algebra with Tests, in: 23rd IEEE Symposium on Logic in Computer Science (LICS'08), IEEE Computer Society, 2008, pp. 105–117.

[46] K. Aboul-Hosn, D. Kozen, Local Variable Scoping and Kleene Algebra with Tests, Journal of Logic and Algebraic Programming 76 (1) (2008) 3–17.

[47] G. Winskel, An Introduction to Event Structures, in: de Bakker et al. [55], pp. 364–397.

[48] T. Hoare, B. Möller, G. Struth, I. Wehrman, Concurrent Kleene Algebra, in: M. Bravetti, G. Zavattaro (Eds.), 20*th* International Conference on Concurrency Theory (CONCUR'09), Vol. 5710 of LNCS, Springer, 2009, pp. 399–414.

[49] J. L. Gischer, Partial Orders and the Axiomatic Theory of Shuffle, Ph.D. thesis, CS Department, Stanford University (1984).

[50] K. R. Abrahamson, Modal Logic of Concurrent Nondeterministic Programs, in: G. Kahn (Ed.), International Sympoisum on Semantics of Concurrent Computation, Vol. 70 of LNCS, Springer, 1979, pp. 21–33.

[51] S. L. Bloom, Z. Ésik, Free Shuffle Algebras in Language Varieties, Theor. Comput. Sci. 163 (1&2) (1996) 55–98.

[52] J. F. Groote, A. Mathijssen, M. A. Reniers, Y. S. Usenko, M. van Weerdenburg, The Formal Specification Language mCRL2, in: E. Brinksma, D. Harel, A. Mader, P. Stevens, R. Wieringa (Eds.), Methods for Modelling

Software Systems (MMOSS'06), Vol. 06351 of Dagstuhl Seminar Proceedings, Internationales Begegnungs- Und Forschungszentrum Fuer Informatik (IBFI), Germany, 2007.

[53] J. F. Groote, A. Ponse, Proof Theory for muCRL: A Language for Processes with Data, in: D. J. Andrews, J. F. Groote, C. A. Middelburg (Eds.), Semantics of Specification Languages, Workshops in Computing, Springer, 1993, pp. 232–251.

[54] J. F. Groote, J. Keiren, A. Mathijssen, B. Ploeger, F. Stappers, C. Tankink, Y. Usenko, M. van Weerdenburg, W. Wesselink, T. Willemse, J. van der Wulp, The mCRL2 Toolset, in: Workshop on Advanced Software Development Tools and Techniques, 2008.

[55] J. W. de Bakker, W. P. de Roever, G. Rozenberg (Eds.), Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, School/Workshop, Noordwijkerhout, The Netherlands, May 30 - June 3, 1988, Proceedings, Vol. 354 of LNCS, Springer, 1989.