Deciding Synchronous Kleene Algebra with Derivatives*

Sabine Broda, Sílvia Cavadas, Miguel Ferreira, and Nelma Moreira

CMUP & DCC, Faculdade de Ciências da Universidade do Porto
Rua do Campo Alegre, 4169-007 Porto, Portugal
sbb@dcc.fc.up.pt,silviacavadas@gmail.com,
miguelferreira108@gmail.com,nam@dcc.fc.up.pt

Abstract. Synchronous Kleene algebra (SKA) is a decidable framework that combines Kleene algebra (KA) with a synchrony model of concurrency. Elements of SKA can be seen as processes taking place within a fixed discrete time frame and that, at each time step, may execute one or more basic actions or then come to a halt. The synchronous Kleene algebra with tests (SKAT) combines SKA with a Boolean algebra. Both algebras were introduced by Prisacariu, who proved the decidability of the equational theory, through a Kleene theorem based on the classical Thompson ε-NFA construction. Using the notion of partial derivatives, we present a new decision procedure for equivalence between SKA terms. The results are extended for SKAT considering automata with transitions labeled by Boolean expressions instead of atoms. This work continous previous research done for KA and KAT, where derivative based methods were used in feasible algorithms for testing terms equivalence.

Keywords: Synchronous Kleene Algebra, Concurrency, Equivalence, Derivative

1 Introduction

Synchronous Kleene algebra (SKA) combines Kleene algebra (KA) with the synchrony model of concurrency of Milner's Synchronous Calculus of Communication Systems (SCCS) [20]. Synchronous here means that two concurrent processes execute a single action simultaneously at each time instant of a unique global clock. Although this synchrony model seems to be a very weak model of concurrency when compared with asynchronous interleaving models, its equational theory is powerful and the SCCS calculus includes the Calculus of Communication Systems (CCS) as a sub-calculus. It also models the Esterel programming language [5], a tool used by the industry [29].

* This work was partially supported by CMUP (UID/MAT/00144/2013), which is funded by FCT (Portugal) with national (MEC) and European structural funds through the programs FEDER, under the partnership agreement PT2020, and through the programme COMPETE and by the Portuguese Government through the FCT under project FCOMP-01-0124-FEDER-020486.

SKA was introduced by Prisacariu [25]. It consists of a KA to which a synchrony operator and a notion of basic action are added. Using a Kleene's style theorem, Prisacariu proved the decidability of the equational theory. He also generalized Kleene algebra with tests (KAT) [15], an equational system that extends Kleene algebra with Boolean algebra. KAT is specially suited to capture and verify properties of simple imperative programs and, in particular, subsumes propositional Hoare logic [16]. For the resulting algebra, called synchronous Kleene algebra with tests (SKAT), the models considered were sets of guarded synchronous strings and decidability was also proved using the so called automata on guarded synchronous strings. SKAT can be seen as an alternative to Hoare logic for reasoning about parallel programs with shared variables in a synchronous system.

Decision procedures for Kleene algebra terms equivalence have been a subject of intense research in recent years [1,12,19,21,7,27,22]. This is partially motivated by the fact that regular expressions can be seen as a program logic that allows to express nondeterministic choice, sequence, and finite iteration of programs. Many proposed procedures decide equivalence based on the computation of a bisimulation (or a bisimulation up-to) between the two expressions [1,21,7,27]. Broda *et al.* studied the average size of derivative based automata both for KA and KAT [9]. For KAT terms, a coalgebraic decision procedure was presented by Kozen [18]. There, derivatives are considered with respect to symbols $\mathtt{v}\sigma$ where σ is an action symbol but $\mathtt{v}$ corresponds to a valuation of the Boolean tests. This induces an exponential blow-up on the number of states or transitions of the automata and an accentuated exponential complexity when testing the equivalence of two KAT expressions (as noted in [23,3]). A. Silva [28] introduced a class of automata over guarded strings that avoids that blow-up. Broda *et al.* studied the average size of some automata of that class [9] and extended finite automata equivalence decision procedures to that class [10]. In this paper we continue this line of work and present new decision procedures for SKA and SKAT equivalence, based on the notion of partial derivatives. For SKA an ε-free NFA construction is presented which leads to smaller automata than the one given by Prisacariu. For SKAT we introduce a class of automata over guarded synchronous strings where transitions are labeled by Boolean expressions instead of valuations. This feature significally improves the performance of the associated methods. For both methods some experimental results are presented and discussed.

2 Deciding Synchronous Kleene Algebra

First we review some concepts related with SKA. A *Kleene algebra* (KA) is an algebraic structure $(\mathcal{A}, +, \cdot, ^*, 0, 1)$, where $+$ and $\cdot$ are binary operations on $\mathcal{A}$, * is a unary operation on $\mathcal{A}$, and 0 and 1 belong to $\mathcal{A}$, such that $(\mathcal{A}, +, \cdot, 0, 1)$ is an idempotent semiring, and * satisfies axioms (10)-(13) below. The natural order $\leq$ in $(\mathcal{A}, +, \cdot, 0, 1)$ is defined by $\alpha \leq \beta$ if and only if $\alpha + \beta = \beta$.

$$1 + \alpha\alpha^* \leq \alpha^* \qquad \alpha + \beta \cdot \gamma \leq \gamma \quad \rightarrow \quad \beta^* \cdot \alpha \leq \gamma$$
$$1 + \alpha^*\alpha \leq \alpha^* \qquad \alpha + \gamma \cdot \beta \leq \gamma \quad \rightarrow \quad \alpha \cdot \beta^* \leq \gamma$$

A *synchronous Kleene algebra* (SKA) over a finite set $\mathsf{A_B}$ is given by a structure $(\mathcal{A}, +, \cdot, \times, ^*, 0, 1, \mathsf{A_B})$, where $\mathsf{A_B} \subseteq \mathcal{A}$, $(\mathcal{A}, +, \cdot, ^*, 0, 1)$ is a Kleene algebra, and $\times$ is a binary operator that is associative, commutative, distributive over $+$, with absorvent element 0 and identity 1. Furthermore, it satisfies $a \times a = a \quad \forall a \in \mathsf{A_B}$, as well as the *synchrony axiom*

$$(\alpha^\times \cdot \alpha) \times (\beta^\times \cdot \beta) = (\alpha^\times \times \beta^\times) \cdot (\alpha \times \beta) \quad \forall \alpha^\times, \beta^\times \in \mathsf{A_B^\times},$$

where the set $\mathsf{A_B^\times}$ is the smallest subset of $\mathcal{A}$ that contains $\mathsf{A_B}$ and is closed for $\times$. As usual, we will omit the operator $\cdot$ whenever it does not give rise to any ambiguity and use the following precedence over the operators: $+ < \cdot < \times < ^*$.

We think of the elements of SKA as processes taking place within a fixed discrete time frame and that, at each time step, may execute one or more basic actions in $\mathsf{A_B}$ or then come to a halt.

The standard model of an SKA over $\mathsf{A_B}$ is the set of languages over the alphabet $\Sigma = \mathcal{P}(\mathsf{A_B}) \setminus \{\emptyset\}$, which we will call *synchronous languages*. Each synchronous language represents a process described by its possible executions, which are given by the words over Σ, each one a sequence of sets of basic actions executed in a single time step. We call $\sigma \in \Sigma$ a *(synchronous) concurrent action*. The *synchronous product* of two words $x = \sigma_1 \cdots \sigma_m$ and $y = \tau_1 \cdots \tau_n$, with $n \geq m$, is defined by

$$x \times y = y \times x = (\sigma_1 \cup \tau_1 \cdots \sigma_m \cup \tau_m)\tau_{m+1} \cdots \tau_n.$$

In particular, the synchronous product of two letters in Σ is their union. The synchronous product of two languages L_1 and L_2 is defined by

$$L_1 \times L_2 = \{\, x \times y \,|\, x \in L_1, y \in L_2 \,\}.$$

It is clear that the synchronous regular languages over $\mathsf{A_B}$ contain the regular languages over Σ. It turns out that they are exactly the same set, i.e., the regular languages over Σ are also closed for $\times$. In [25], the classical Thompson construction for regular languages [30] is extended to build an automaton accepting the synchronous product of two languages given by their automata.

We now introduce the SKA analogue of the regular expressions. We denote by $\mathcal{T}_{\mathsf{SKA}}$ the set of SKA terms, containing 0 plus all terms generated by the grammar

$$\alpha \to 1 \mid a \mid \alpha + \alpha \mid \alpha \cdot \alpha \mid \alpha \times \alpha \mid \alpha^* \quad (a \in \mathsf{A_B}). \tag{1}$$

Note that we do not include in $\mathcal{T}_{\mathsf{SKA}}$ compound expressions that have 0 as a subexpression. Given $\alpha \in \mathcal{T}_{\mathsf{SKA}}$, the language $\mathcal{L}(\alpha)$ denoted by α is inductively defined as follows, $\mathcal{L}(a) = \{\{a\}\}$, $\mathcal{L}(0) = \emptyset$, $\mathcal{L}(1) = \{\varepsilon\}$, $\mathcal{L}(\alpha^*) = \mathcal{L}(\alpha)^*$, $\mathcal{L}(\alpha + \beta) = \mathcal{L}(\alpha) \cup \mathcal{L}(\beta)$, $\mathcal{L}(\alpha\beta) = \mathcal{L}(\alpha)\mathcal{L}(\beta)$, $\mathcal{L}(\alpha \times \beta) = \mathcal{L}(\alpha) \times \mathcal{L}(\beta)$.

Example 1. Let $\mathsf{A_B} = \{a, b\}$, hence $\Sigma = \{\{a\}, \{b\}, \{a, b\}\}$, and consider the SKA term $\alpha = (a(b + a)^*) \times (a + bb)^*$ over $\mathsf{A_B}$. Then

$$\begin{aligned}\mathcal{L}(\alpha) &= \{\{a\}, \{a\}\{a\}, \{a\}\{b\}, \ldots\} \times \{\varepsilon, \{a\}, \{a\}\{a\}, \{b\}\{b\}, \ldots\} \\ &= \{\{a\}, \{a\}\{a\}, \{a\}\{b\}, \{a\}\{a, b\}, \{a, b\}\{b\}, \{a, b\}\{a, b\}, \ldots\}.\end{aligned}$$

Given $\alpha, \beta \in \mathcal{T}_{\mathsf{SKA}}$, we say that they are *equivalent* if they denote the same language, i.e., $\mathcal{L}(\alpha) = \mathcal{L}(\beta)$. We also define $\varepsilon(\alpha) = 1$ if $\varepsilon \in \mathcal{L}(\alpha)$, and $\varepsilon(\alpha) = 0$ otherwise. A recursive definition of $\varepsilon : \mathcal{T}_{\mathsf{SKA}} \longrightarrow \{0, 1\}$ is given by the following, $\varepsilon(a) = \varepsilon(0) = 0$, $\varepsilon(1) = \varepsilon(\alpha^*) = 1$, $\varepsilon(\alpha + \beta) = \varepsilon(\alpha) + \varepsilon(\beta)$, and $\varepsilon(\alpha\beta) = \varepsilon(\alpha \times \beta) = \varepsilon(\alpha) \cdot \varepsilon(\beta)$. We generalize ε for sets $S \subseteq \mathcal{T}_{\mathsf{SKA}}$ by $\varepsilon(S) = \sum_{\alpha \in S} \varepsilon(\alpha)$.

2.1 Partial Derivative Automata for SKA

A *nondeterministic finite automaton* (NFA) is a tuple $\mathcal{A} = \langle S, \Sigma, S_0, \delta, F \rangle$, where S is a finite set of states, Σ is a finite alphabet, $S_0 \subseteq S$ a set of initial states, $\delta : S \times \Sigma \longrightarrow \mathcal{P}(S)$ the transition function, and $F \subseteq S$ a set of final states. The transition function δ is extended to words and sets of states in the natural way. A word $x \in \Sigma^*$ is *accepted* by $\mathcal{A}$ if and only if $\delta(S_0, x) \cap F \neq \emptyset$. The *language of* $\mathcal{A}$ is the set of words accepted by $\mathcal{A}$ and denoted by $\mathcal{L}(\mathcal{A})$.

In the context of SKA, we consider the alphabet $\Sigma = \mathcal{P}(\mathsf{A_B}) \setminus \{\emptyset\}$ and call the NFA a *nondeterministic automaton on synchronous strings*. Prisacariu presented a method of converting an SKA expression into an equivalent ε-NFA (in an ε-NFA transitions may be labelled by ε), based on the classical Thompson construction. Due to the local behaviour of the synchronization operator, in each step it is necessary to eliminate all ε-transitions except those entering the final state. The step for the synchronous product $\alpha \times \beta$ involves the construction of a classic product automaton from the automata corresponding to α and β, respectively. This leads easily to large automata for relatively small expressions. We present now a new method of converting of an SKA expression into an equivalent ε-free NFA. This method extends the classical partial derivative automata construction for regular expressions [4] and provides a new proof that the set of synchronous regular languages over $\mathsf{A_B}$ is precisely the set of regular languages over Σ.

As usual, the *left-quotient* of a synchronous language L w.r.t. a synchronous concurrent action σ is the set $\sigma^{-1}L = \{\ x \mid \sigma x \in L\ \}$. The left quotient of L w.r.t. a word $x \in \Sigma^*$ is inductively defined by $\varepsilon^{-1}L = L$ and $(x\sigma)^{-1}L = \sigma^{-1}(x^{-1}L)$. Antimirov [4] introduced the notion of partial derivatives which we now generalize to the set $\mathcal{T}_{\mathsf{SKA}}$. Given sets $S, T \subseteq \mathcal{T}_{\mathsf{SKA}}$, let $S \odot T = \{\ \alpha\beta \mid \alpha \in S\backslash\{0\}, \beta \in T\backslash\{0\}\ \}$ and $S \otimes T = \{\ \alpha \times \beta \mid \alpha \in S\backslash\{0\}, \beta \in T\backslash\{0\}\ \}$. We consider $\alpha \odot S = \{\alpha\} \odot S$, and similarly for $S \odot \alpha$, $\alpha \otimes S$ and $S \otimes \alpha$. These definitions serve the following.

Definition 2. *The set of partial derivatives of a term $\alpha \in \mathcal{T}_{\mathsf{SKA}}$ w.r.t. the letter $\sigma \in \Sigma$, denoted by $\partial_\sigma(\alpha)$, is inductively defined by*

$$\partial_\sigma(0) = \partial_\sigma(1) = \emptyset \qquad \partial_\sigma(\alpha^*) = \partial_\sigma(\alpha) \odot \alpha^*$$

$$\partial_\sigma(a) = \begin{cases} \{1\} \text{ if } \sigma = \{a\} \\ \emptyset \text{ otherwise} \end{cases} \qquad \begin{array}{l} \partial_\sigma(\alpha + \beta) = \partial_\sigma(\alpha) \cup \partial_\sigma(\beta) \\ \partial_\sigma(\alpha\beta) = \partial_\sigma(\alpha) \odot \beta \ \cup\ \varepsilon(\alpha) \odot \partial_\sigma(\beta) \end{array}$$

$$\partial_\sigma(\alpha \times \beta) = \left(\bigcup_{\sigma_1 \times \sigma_2 = \sigma} \partial_{\sigma_1}(\alpha) \otimes \partial_{\sigma_2}(\beta)\right) \ \cup\ \varepsilon(\alpha) \otimes \partial_\sigma(\beta) \ \cup\ \varepsilon(\beta) \otimes \partial_\sigma(\alpha).$$

The set of partial derivatives of $\alpha \in \mathcal{T}_{\mathsf{SKA}}$ w.r.t. a word $x \in \Sigma^*$ is inductively defined by $\partial_\varepsilon(\alpha) = \{\alpha\}$ and $\partial_{x\sigma}(\alpha) = \partial_\sigma(\partial_x(\alpha))$, where, given a set $S \subseteq \mathcal{T}_{\mathsf{SKA}}$, $\partial_\sigma(S) = \bigcup_{\alpha \in S} \partial_\sigma(\alpha)$.

We denote by $\partial(\alpha)$ the set of all partial derivatives of α, $\partial(\alpha) = \bigcup_{x\in\Sigma^*} \partial_x(\alpha)$, and by $\partial^+(\alpha)$ the set of partial derivatives excluding the trivial derivative by ε, $\partial^+(\alpha) = \bigcup_{x\in\Sigma^+} \partial_x(\alpha)$. Given a set $S \subseteq \mathcal{T}_{\mathsf{SKA}}$, we define $\mathcal{L}(S) = \bigcup_{\alpha\in S} \mathcal{L}(\alpha)$. It is straightforward to show that for every $\mathcal{T}_{\mathsf{SKA}}$ term α and word x, $\mathcal{L}(\partial_x(\alpha)) = x^{-1}\mathcal{L}(\alpha)$. The following lemma will be used to show that $\partial(\alpha)$ is finite, as in the case for standard regular expressions.

Lemma 3. *The set $\partial^+(\alpha)$ satisfies the following.*

$$\begin{array}{ll} \partial^+(0) = \partial^+(1) = \emptyset & \partial^+(\alpha+\beta) \subseteq \partial^+(\alpha) \cup \partial^+(\beta) \\ \partial^+(a) = \{1\} \quad (a \in \mathsf{A_B}) & \partial^+(\alpha\beta) \subseteq \partial^+(\alpha) \odot \beta \cup \partial^+(\beta) \\ \partial^+(\alpha^*) \subseteq \partial^+(\alpha) \odot \alpha^* & \partial^+(\alpha \times \beta) \subseteq \partial^+(\alpha) \otimes \partial^+(\beta) \cup \partial^+(\alpha) \cup \partial^+(\beta). \end{array}$$

Proof. The proof proceeds by induction on the structure of α. It is clear that for $\partial^+(0)$, $\partial^+(1)$ and, for $\partial^+(a)$, $a \in \mathsf{A_B}$, the result is true. Now, suppose the claim is true for α and β, with $|\alpha|_{\mathsf{A_B}} \neq 0$ and $|\beta|_{\mathsf{A_B}} \neq 0$. Otherwise, one has $|\partial^+(\alpha)| = 0$ and/or $|\partial^+(\beta)| = 0$, simplifying the arguments below. There are four induction cases to consider, in which we will make use of the fact that, for any SKA expression γ and letter $\sigma \in \Sigma$, the set $\partial^+(\gamma)$ is closed for taking derivatives w.r.t. σ, i.e., $\partial_\sigma(\partial^+(\gamma)) \subseteq \partial^+(\gamma)$.

i. One can check by induction on the length of x that, for $x \in \Sigma^+$, $\partial_x(\alpha+\beta) = \partial_x(\alpha) \cup \partial_x(\beta)$. Hence, $\partial^+(\alpha+\beta) = \partial^+(\alpha) \cup \partial^+(\beta)$.
ii. We will prove by induction on the length of x that $\partial_x(\alpha\beta) \subseteq \partial^+(\alpha) \odot \beta \cup \partial^+(\beta)$ for every word $x \in \Sigma^+$. The claim is true for $\sigma \in \Sigma$ since $\partial_\sigma(\alpha\beta) = \partial_\sigma(\alpha) \odot \beta \cup \varepsilon(\alpha) \odot \partial_\sigma(\beta)$. Assuming it is true for x, $\partial_{x\sigma}(\alpha\beta) = \partial_\sigma(\partial_x(\alpha\beta)) \subseteq \partial_\sigma(\partial^+(\alpha)\odot\beta\cup\partial^+(\beta)) \subseteq \partial_\sigma(\partial^+(\alpha))\odot\beta\cup\partial_\sigma(\beta)\cup\partial_\sigma(\partial^+(\beta)) \subseteq \partial^+(\alpha) \odot \beta \cup \partial^+(\beta)$.
iii. We prove by induction on the length of x that, for every word $x \in \Sigma^+$, $\partial_x(\alpha \times \beta) \subseteq \partial^+(\alpha) \otimes \partial^+(\beta) \cup \partial^+(\alpha) \cup \partial^+(\beta)$. The claim is true for $\sigma \in \Sigma$ because $\partial_\sigma(\alpha\times\beta) = \bigcup_{\sigma_1\times\sigma_2=\sigma} \partial_{\sigma_1}(\alpha)\otimes\partial_{\sigma_2}(\beta)\cup\varepsilon(\alpha)\otimes\partial_\sigma(\beta)\cup\varepsilon(\beta)\otimes\partial_\sigma(\alpha)$; supposing it is true for x, $\partial_{x\sigma}(\alpha \times \beta) = \partial_\sigma(\partial_x(\alpha \times \beta)) \subseteq \partial_\sigma(\partial^+(\alpha) \otimes \partial^+(\beta)\cup\partial^+(\alpha)\cup\partial^+(\beta)) \subseteq (\bigcup_{\sigma_1\times\sigma_2=\sigma} \partial_{\sigma_1}(\partial^+(\alpha))\otimes\partial_{\sigma_2}(\partial^+(\beta)))\cup\partial_\sigma(\partial^+(\alpha))\cup \partial_\sigma(\partial^+(\beta)) \subseteq \partial^+(\alpha) \otimes \partial^+(\beta) \cup \partial^+(\alpha) \cup \partial^+(\beta)$.
iv. We show by induction on the length of x that $\partial_x(\alpha^*) \subseteq \partial^+(\alpha) \odot \alpha^*$ for $x \in \Sigma^+$. It is true for $\sigma \in \Sigma$ because $\partial_\sigma(\alpha^*) = \partial_\sigma(\alpha) \odot \alpha^*$; supposing the claim true for x, $\partial_{\sigma x}(\alpha^*) = \partial_\sigma(\partial_x(\alpha^*)) \subseteq \partial_\sigma(\partial^+(\alpha) \odot \alpha^*) \subseteq \partial_\sigma(\partial^+(\alpha)) \odot \alpha^* \cup \partial_\sigma(\alpha^*) \subseteq \partial^+(\alpha) \odot \alpha^* \cup \partial_\sigma(\alpha) \odot \alpha^* \subseteq \partial^+(\alpha) \odot \alpha^*$. □

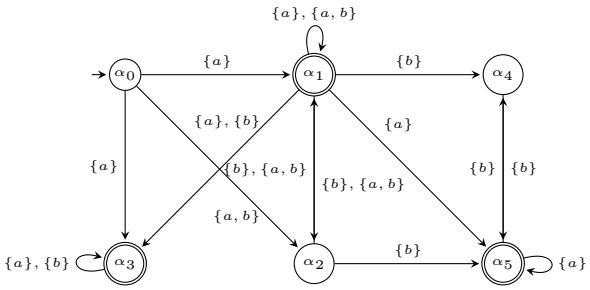
Now, it is easy to obtain the following upper bound for the size of $\partial^+(\alpha)$.

Proposition 4. *Given $\alpha \in \mathcal{T}_{\mathsf{SKA}}$, $|\partial^+(\alpha)| \leq 2^{|\alpha|_{\mathsf{A_B}}} - 1$, where $|\alpha|_{\mathsf{A_B}}$ denotes the number of occurrences of elements of $\mathsf{A_B}$ in α. Thus, $|\partial(\alpha)| \leq 2^{|\alpha|_{\mathsf{A_B}}}$.*

We note that this upper bound is exactly the same obtained for the number of partial derivatives for regular expressions with the shuffle operator [11]. In the latter case, however, the correspondent version of Lemma 3 establishes equalities instead of inclusions.

We extend to SKA terms the standard Antimirov automaton or partial derivative automaton. Given $\alpha \in \mathcal{T}_{\mathsf{SKA}}$, we define the *partial derivative automaton* associated to α by $\mathcal{A}(\alpha) = \langle \partial(\alpha), \Sigma, \{\alpha\}, \delta_\alpha, F_\alpha \rangle$, where $F_\alpha = \{\ \gamma \in \partial(\alpha) \mid \varepsilon(\gamma) = 1\ \}$ and $\delta_\alpha(\gamma, \sigma) = \partial_\sigma(\gamma)$. Then, it is easy to see that $\mathcal{L}(\mathcal{A}(\alpha)) = \mathcal{L}(\alpha)$.

Example 5. Consider again the expression α from Example 1 and let $\beta = (b + a)^*$ and $\gamma = (a + bb)^*$, i.e. $\alpha = (a\beta) \times \gamma$. Furthermore, let $\alpha_0 = \alpha$, $\alpha_1 = \beta \times \gamma$, $\alpha_2 = \beta \times (b\gamma)$, $\alpha_3 = \beta$, $\alpha_4 = b\gamma$, and $\alpha_5 = \gamma$. The nonempty sets of partial derivatives of α are the following: $\partial_{\{a\}}(\alpha_0) = \{\alpha_1, \alpha_3\}$, $\partial_{\{a,b\}}(\alpha_0) = \{\alpha_2\}$, $\partial_{\{a\}}(\alpha_1) = \{\alpha_1, \alpha_3, \alpha_5\}$, $\partial_{\{b\}}(\alpha_1) = \{\alpha_2, \alpha_3, \alpha_4\}$, $\partial_{\{a,b\}}(\alpha_1) = \{\alpha_1, \alpha_2\}$, $\partial_{\{b\}}(\alpha_2) = \{\alpha_1, \alpha_5\}$, $\partial_{\{a,b\}}(\alpha_2) = \{\alpha_1\}$, $\partial_{\{a\}}(\alpha_3) = \partial_{\{b\}}(\alpha_3) = \{\alpha_3\}$, $\partial_{\{b\}}(\alpha_4) = \partial_{\{a\}}(\alpha_5) = \{\alpha_5\}$, $\partial_{\{b\}}(\alpha_5) = \{\alpha_4\}$. Then, $\mathcal{A}(\alpha)$ is the following.



It is worthwhile to note that this automaton has 6 states and 19 transitions, while the one obtained using Prisacariu's Thompson-based construction has 16 states and 73 transitions, even after some necessary ε-transition eliminations.

2.2 Equivalence of SKA Expressions

We are interested in an algorithm that decides whether or not two SKA terms represent the same regular language. Since we already know how to construct an NFA that accepts a given SKA term, the problem is tantamount to deciding the language equivalence of two automata. One possible approach is to search for the existence of a bisimulation in the determinized NFAs (DFAs), as presented by Hopcroft and Karp [14]. This algorithm can be easily extended to NFAs as in Almeida *et al.* [1]. A presentation of this algorithm and an improved variant, together with proofs of correctness, can be found in Bonchi and Pous [6].

2.3 Implementation and Experimental Results

A Python module for manipulating SKA terms and automata over synchronous strings was implemented within the FAdo library [26], which includes several algorithms for regular expressions and finite automata. For the efficient computation of the set of partial derivatives of a term w.r.t. a symbol, in FAdo a function is used, that given an expression α computes the set of pairs $(\sigma, \partial_\sigma(\alpha))$

with $\sigma \in \Sigma$ [4]. We extended it to the synchronous product,

$$f : \mathcal{T}_{\mathsf{SKA}} \rightarrow \mathcal{P}(\Sigma \times \mathcal{P}(\mathcal{T}_{\mathsf{SKA}}))$$
$$f(0) = f(1) = \emptyset \qquad f(a) = \{(\{a\}, \varepsilon)\} \quad f(\alpha^*) = f(\alpha) \cdot \alpha^*$$
$$f(\alpha + \beta) = f(\alpha) \cup f(\beta) \qquad f(\alpha\beta) = f(\alpha) \odot \beta \cup f(\beta) \odot \varepsilon(\alpha)$$
$$f(\alpha \times \beta) = \{\ (\sigma_1 \cup \sigma_2, \alpha_1 \times \alpha_2) \mid (\sigma_1, \alpha_1) \in f(\alpha), (\sigma_2, \alpha_2) \in f(\beta)\ \}$$
$$\cup\, f(\beta) \odot \varepsilon(\alpha) \cup f(\alpha) \odot \varepsilon(\beta)$$

where, as before, for $\alpha \neq 1$, $\Gamma \odot \alpha = \{\ (\sigma, \alpha'\alpha) \mid (\sigma, \alpha') \in \Gamma\ \}$.

For running some experiments we uniformly random generated SKA terms. The FAdo random generator has as input a grammar, the size k of the alphabet, and the size n of the words to be generated. A prefix notation version of the grammar (1) was used in order to obtain terms, uniformly generated in the size $|\alpha|$ of the syntactic tree (i.e. parentheses not counted). For each size, $n = |\alpha|$ and $k = |\mathsf{A_B}|$, samples of 1000 terms were generated. We compared the sizes of the partial derivative automata $\mathcal{A}(\alpha)$ and the automata proposed by Prisacariu, a variant of the Thompson construction, $(S_{\text{tho}}, \Sigma, I_{\text{tho}}, \delta_{\text{tho}}, F_{\text{tho}})$. Table 1 presents average values obtained for $n \in \{50, 100\}$ and $k \in \{5, 10, 20\}$.

k	$\lvert\alpha\rvert$	$\lvert S_{\text{tho}}\rvert$	$\lvert\delta_{\text{tho}}\rvert$	$\lvert\partial(\alpha)\rvert$	$\lvert\delta_\alpha\rvert$	$\frac{\lvert\partial(\alpha)\rvert}{\lvert S_{\text{tho}}\rvert}$	$\frac{\lvert\delta_\alpha\rvert}{\lvert\delta_{\text{tho}}\rvert}$
5	50	59	496	23	159	0.389	0.321
5	100	491	47288	128	4133	0.261	0.087
10	50	49	271	18	97	0.364	0.358
10	100	358	15096	96	1691	0.268	0.112
20	50	44	165	16	69	0.364	0.418
20	100	194	2126	60	559	0.309	0.263

Table 1. Experimental results for uniform random generated $\mathcal{T}_{\mathsf{SKA}}$ expressions

Analyzing the table, it seems that the partial derivative automaton is always smaller than the Thompson-like construction, and that the exponential blow up of the automaton size may not occur on average. For regular expressions it is known that after eliminating ε-transitions from the Thompson automaton one obtains the Glushkov automaton [13], of which the partial derivative automaton is a quotient. Asymptotically and on average the size of the partial derivative automaton is half the size of the Glushkov automaton [8], which on the other hand is linear on the size of the expression. As noticed before, for the synchronous product the Thompson construction considers a product automaton and thus a quadratic number of transitions is expected. We also note that for every synchronisation ε-transitions are eliminated, reducing the size of the resulting automata that otherwise should be much larger. No such procedures are needed for the

partial derivative automata. For testing the equivalence of SKA terms we can use one of the algorithms mentioned above.

3 Deciding Synchronous Kleene Algebra with Tests

Synchronous Kleene algebra with tests (SKAT) was also introduced by Prisacariu as a natural extension of the Kleene algebra with tests to the synchronous setting. The SKA axiomatization was extended to SKAT, whose standard models are sets of guarded synchronous strings. Prisacariu defined automata over guarded synchronous strings that were based on the ones considered by Kozen for guarded strings [17]. In the synchronous case, automata were built in two layers: one that processed a synchronous string and another to represent the valuations of the boolean tests (called atoms, as defined below). Our contribution in this section is to consider a much simpler notion of automata and to show that the derivative based methods developed in the previous section for SKA can be extended to SKAT. We use standard finite automata where transitions are labeled both with action symbols and boolean tests (instead of atoms). This kind of automata for KAT terms were introduced by Silva [28] and Broda *et al.* [9,10]. In the next subsection, we revise the notions of SKAT and guarded synchronous strings.

3.1 SKAT and Guarded Synchronous Strings

Formally, an SKAT is a structure $(\mathcal{A}, \mathcal{B}, +, \cdot, \times, ^*, \neg, 0, 1, \mathsf{A_B}, \mathsf{T})$, where $\mathsf{T} \subseteq \mathcal{B} \subseteq \mathcal{A}$ and $\mathsf{A_B}$ and T are disjoint finite sets, $(\mathcal{A}, +, \cdot, \times, ^*, 0, 1, \mathsf{A_B} \cup \mathsf{T})$ is an SKA, $(\mathcal{B}, +, \cdot, \neg, 0, 1)$ and $(\mathcal{B}, +, \times, \neg, 0, 1)$ are Boolean algebras, and $(\mathcal{B}, +, \cdot, \times, 0, 1)$ is a subalgebra of $(\mathcal{A}, +, \cdot, \times, 0, 1)$.

Similar to what was done for SKA, we consider the set $\mathcal{B}_{\mathsf{SKAT}}$ of boolean expressions and the set $\mathcal{T}_{\mathsf{SKAT}}$ of SKAT expressions over $\mathsf{A_B} \cup \mathsf{T}$. $\mathcal{B}_{\mathsf{SKAT}}$ is the set of terms finitely generated from $\mathsf{T} \cup \{0, 1\}$ and operators $+, \cdot, \times, \neg$, while $\mathcal{T}_{\mathsf{SKAT}}$ denotes the set of terms finitely generated from $\mathsf{A_B} \cup \mathcal{B}_{\mathsf{SKAT}}$ and operators $+, \cdot, \times, ^*$. Elements of $\mathcal{B}_{\mathsf{SKAT}}$ and $\mathcal{T}_{\mathsf{SKAT}}$ will be denoted by $b, b_1, \ldots$ and $\alpha, \beta, \alpha_1, \ldots$, respectively, and are generated by the following grammar

$$b \to 0 \mid 1 \mid t \mid b + b \mid b \cdot b \mid b \times b \mid \neg b \quad (t \in \mathsf{T}),$$
$$\alpha \to a \mid b \mid \alpha + \alpha \mid \alpha \cdot \alpha \mid \alpha \times \alpha \mid \alpha^* \quad (a \in \mathsf{A_B}).$$

The set At of *atoms* over $\mathsf{T} = \{t_0, \ldots, t_{l-1}\}$, with $l \geq 1$, is the set of all boolean assignments to all elements of T, i.e. $\mathsf{At} = \{\ x_0 \cdots x_{l-1} \mid x_i \in \{t_i, \bar{t}_i\},\ t_i \in \mathsf{T}\ \}$. We denote elements of At by $\mathtt{v}$, $\mathtt{v}_1$, etc. Note that each atom $\mathtt{v} \in \mathsf{At}$ has associated a binary word of l bits $(w_0 \cdots w_{l-1})$ where $w_i = 0$ if $\bar{t}_i \in \mathtt{v}$, and $w_i = 1$ if $t_i \in \mathtt{v}$. The standard model of SKAT consists of the sets of guarded synchronous strings. The set of guarded synchronous strings over $\mathsf{A_B} \cup \mathsf{T}$ is $\mathsf{GSS} = (\mathsf{At} \cdot \Sigma)^* \cdot \mathsf{At}$, where, as before, $\Sigma = \mathcal{P}(\mathsf{A_B}) \setminus \{\emptyset\}$. For $x = \mathtt{v}_0\sigma_1 \cdots \sigma_m\mathtt{v}_m$ and $y = \mathtt{v}'_0\sigma'_1 \cdots \sigma'_n\mathtt{v}'_n \in \mathsf{GSS}$, where $m, n \geq 0$, $\mathtt{v}_i, \mathtt{v}'_j \in \mathsf{At}$ and $\sigma_i, \sigma'_j \in \Sigma$, we define the *fusion product* $x \diamond y = \mathtt{v}_0\sigma_1 \cdots \sigma_m\mathtt{v}_m\sigma'_1 \cdots \sigma'_n\mathtt{v}'_n$, if $\mathtt{v}_m = \mathtt{v}'_0$, leaving it

undefined otherwise. Similarly, for $m \leq n$ the product $x \times y = y \times x$ is defined only if $\mathtt{v}_0 = \mathtt{v}'_0, \ldots, \mathtt{v}_m = \mathtt{v}'_m$ by $x \times y = \mathtt{v}_0(\sigma_1 \cup \sigma'_1) \cdots (\sigma_m \cup \sigma'_m)\mathtt{v}_m\sigma'_{m+1} \cdots \sigma'_n\mathtt{v}'_n$. For sets $X, Y \subseteq \mathsf{GSS}$, $X \diamond Y = \{\ x \diamond y \mid x \in X,\ y \in Y,\ x \diamond y \text{ exists}\ \}$ and $X \times Y = \{\ x \times y \mid x \in X,\ y \in Y,\ x \times y \text{ exists}\ \}$. Finally, let $X^0 = \mathsf{At}$ and $X^{n+1} = X \diamond X^n$, for $n \geq 0$, and define $X^* = \bigcup_{n \geq 0} X^n$.

Given a SKAT expression α, we define $\mathsf{GSS}(\alpha) \subseteq \mathsf{GSS}$ inductively as follows,

$$\begin{array}{lll} \mathsf{GSS}(a) & = \{\ \mathtt{v}_1\{a\}\mathtt{v}_2 \mid \mathtt{v}_1, \mathtt{v}_2 \in \mathsf{At}\ \} & \mathsf{GSS}(\alpha \cdot \beta) = \mathsf{GSS}(\alpha) \diamond \mathsf{GSS}(\beta) \\ \mathsf{GSS}(b) & = \{\ \mathtt{v} \mid \mathtt{v} \in \mathsf{At} \wedge \mathtt{v} \leq b\ \} & \mathsf{GSS}(\alpha \times \beta) = \mathsf{GSS}(\alpha) \times \mathsf{GSS}(\beta) \\ \mathsf{GSS}(\alpha + \beta) & = \mathsf{GSS}(\alpha) \cup \mathsf{GSS}(\beta) & \mathsf{GSS}(\alpha^*) = \mathsf{GSS}(\alpha)^*, \end{array}$$

where $\mathtt{v} \leq b$ if $\mathtt{v} \to b$ is a propositional tautology. For $T \subseteq \mathcal{T}_{\mathsf{SKAT}}$, let $\mathsf{GSS}(T) = \bigcup_{\alpha \in T} \mathsf{GSS}(\alpha)$. Given two $\mathcal{T}_{\mathsf{SKAT}}$ expressions α and β, we say that they are *equivalent* if $\mathsf{GSS}(\alpha) = \mathsf{GSS}(\beta)$.

3.2 Automata for Guarded Synchronous Strings

We extend to for guarded synchronous strings the automata defined for KAT in [28,9,10]. Besides their simplicity when compared with the two-level automata of Prisacariu, their transitions are labeled with tests instead of atoms, avoiding in this way the inevitable exponential blow-up on the size of the automata induced by the number of valuations of tests.

A *(nondeterministic) automaton with tests* (NTA) over the alphabets Σ and T is a tuple $\mathcal{A} = \langle S, s_0, o, \delta \rangle$, where S is a finite set of states, $s_0 \in S$ is the initial state, $o : S \to \mathcal{B}_{\mathsf{SKAT}}$ is the output function, and $\delta \subseteq \mathcal{P}(S \times (\mathcal{B}_{\mathsf{SKAT}} \times \Sigma) \times S)$ is the transition relation. A synchronous guarded string $\mathtt{v}_0\sigma_1 \ldots \sigma_n\mathtt{v}_n$, with $n \geq 0$, is accepted by the automaton $\mathcal{A}$ if and only if there is a sequence of states $s_0, s_1, \ldots, s_n \in S$, where s_0 is the initial state, and, for $i = 0, \ldots, n-1$, one has $\mathtt{v}_i \leq b_i$ for some $(s_i, (b_i, \sigma_{i+1}), s_{i+1}) \in \delta$, and $\mathtt{v}_n \leq o(s_n)$. The set of all guarded strings accepted by $\mathcal{A}$ is denoted by $\mathsf{GSS}(\mathcal{A})$. We say that an SKAT expression $\alpha \in \mathcal{T}_{\mathsf{SKAT}}$ is *equivalent* to an automaton $\mathcal{A}$, and write $\alpha = \mathcal{A}$, if $\mathsf{GSS}(\mathcal{A}) = \mathsf{GSS}(\alpha)$.

3.3 Partial Derivatives for SKAT

In the following, we extend the notion of partial derivative, previously defined in [10] for KAT, to SKAT expressions. The main novelty of the approach in [10] is that derivatives are considered only w.r.t. action symbols σ instead of all combinations $\mathtt{v}\sigma$ for $\mathtt{v} \in \mathsf{At}$ and $\sigma \in \Sigma$.

Definition 6. *For $\alpha \in \mathcal{T}_{SKAT}$ and $\sigma \in \Sigma$, the set $\partial_\sigma(\alpha)$ of* partial derivatives *of α w.r.t. σ is a subset of $\mathcal{B}_{SKAT} \times \mathcal{T}_{SKAT}$ inductively defined as follows,*

$$\partial_\sigma(a) = \begin{cases} \{(1,1)\} & \text{if } \sigma = \{a\} \\ \emptyset & \text{otherwise} \end{cases} \qquad \begin{array}{ll} \partial_\sigma(\alpha^*) & = \partial_\sigma(\alpha) \odot \alpha^* \\ \partial_\sigma(\alpha + \beta) & = \partial_\sigma(\alpha) \cup \partial_\sigma(\beta) \\ \partial_\sigma(\alpha\beta) & = \partial_\sigma(\alpha) \odot \beta \cup \mathsf{out}(\alpha) \odot \partial_\sigma(\beta) \end{array}$$

$$\partial_\sigma(b) = \emptyset$$

$$\partial_\sigma(\alpha \times \beta) = \big(\textstyle\bigcup_{\sigma_1 \times \sigma_2 = \sigma} \partial_{\sigma_1}(\alpha) \otimes \partial_{\sigma_2}(\beta)\big) \cup \mathsf{out}(\alpha) \otimes \partial_\sigma(\beta) \cup \mathsf{out}(\beta) \otimes \partial_\sigma(\alpha),$$

where $\mathsf{out} : \mathcal{T}_{SKAT} \longrightarrow \mathcal{B}_{SKAT}$ *is defined by*

$$\begin{array}{ll} \mathsf{out}(a) = 0 \quad \mathsf{out}(b) = b \quad \mathsf{out}(\alpha^*) = 1 & \mathsf{out}(\alpha + \beta) = \mathsf{out}(\alpha) + \mathsf{out}(\beta) \\ \mathsf{out}(\alpha \cdot \beta) = \mathsf{out}(\alpha) \cdot \mathsf{out}(\beta) & \mathsf{out}(\alpha \times \beta) = \mathsf{out}(\alpha) \times \mathsf{out}(\beta), \end{array}$$

and for $S, T \subseteq \mathcal{B}_{SKAT} \times \mathcal{T}_{SKAT}$, $\alpha \neq 0$ *in* $\mathcal{T}_{SKAT}$, *and* $b \neq 0$ *in* $\mathcal{B}_{SKAT}$, $S \odot \alpha = \{\ (b', \alpha'\alpha) \mid (b', \alpha') \in S, \alpha' \neq 0\ \}$, $b \odot S = \{\ (b \cdot b', \alpha') \mid (b', \alpha') \in S, b' \neq 0\ \}$, $S \odot 0 = 0 \odot S = \emptyset$ *and* $S \otimes T = \{\ (b \times b', \alpha \times \alpha') \mid (b, \alpha) \in S, (b', \alpha') \in T, b, b', \alpha, \alpha' \neq 0\ \}$. *Given* $\alpha \in \mathcal{T}_{SKAT}$ *and* $\sigma \in \Sigma$ *we define the set of expressions derived from* α *w.r.t. a letter* σ *by* $\Delta_\sigma(\alpha) = \{\ \alpha' \mid (b, \alpha') \in \partial_\sigma(\alpha)$ *for some* $b\ \}$. *The functions* ∂_σ, out, *and* Δ_σ *are naturally extended to sets of SKAT expressions and words* $\in \Sigma^\star$.

Let $\Delta(\alpha) = \bigcup_{x \in \Sigma^*} \Delta_x(\alpha)$. Given $\alpha \in \mathcal{T}_{\mathsf{SKAT}}$, we define the partial derivative automaton associated to α by $\mathcal{A}(\alpha) = \langle \Delta(\alpha), \alpha, \mathsf{out}, \delta_\alpha \rangle$, where

$$\delta_\alpha = \{\ (\gamma, (b, \sigma), \gamma') \mid \gamma \in \Delta(\alpha), (b, \gamma') \in \partial_\sigma(\gamma)\ \}.$$

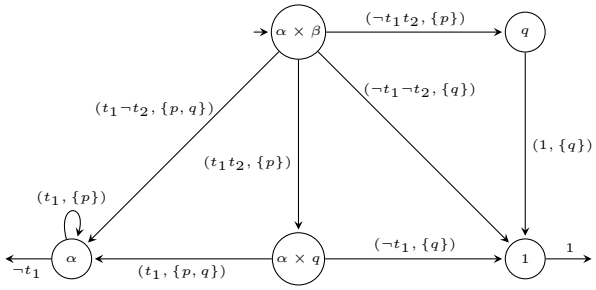
In order to justify the correctness of the partial derivative automaton, i.e., to show that $\mathsf{GSS}(\mathcal{A}(\alpha)) = \mathsf{GSS}(\alpha)$, we first note that, using an almost identical proof as for Proposition 4 in Section 2, one can show by induction on the structure of $\alpha \in \mathcal{T}_{\mathsf{SKAT}}$ that $|\Delta^+(\alpha)| \leq 2^{|\alpha|_{\mathsf{A_B}}} - 1$, where again $\Delta^+(\alpha)$ is the set of expressions derived from α excluding the trivial derivation w.r.t. the empty word ε. Thus, $\Delta(\alpha)$ is finite. Finally, the correctness of the partial derivative automaton is guaranteed by the following result.

Proposition 7. *Let* $\gamma \in SKAT$ *and* $x \in (\mathsf{At} \times \Sigma)^* \cdot \mathsf{At}$. *If* $x = \mathtt{v}$, *then* $x \in \mathsf{GSS}(\gamma)$ *if and only if* $\mathtt{v} \leq \mathsf{out}(\gamma)$. *Furthermore, if* $x = \mathtt{v}\sigma x'$, *then* $x \in \mathsf{GSS}(\gamma)$ *if and only if there is some* $(b, \gamma') \in \partial_\sigma(\gamma)$, *such that* $\mathtt{v} \leq b$ *and* $x' \in \mathsf{GSS}(\gamma')$.

Proof. The proof is by induction on the structure of γ. We only present for *ii.* the cases for $\gamma = \alpha\beta$ and $\gamma = \alpha \times \beta$. Let $\gamma = \alpha\beta$ and $x = \mathtt{v}\sigma x'$. One has $x \in \mathsf{GSS}(\alpha\beta)$ iff $x \in \mathsf{GSS}(\alpha) \diamond \mathsf{GSS}(\beta)$. This means that either, $\mathtt{v} \in \mathsf{GSS}(\alpha)$ and $x \in \mathsf{GSS}(\beta)$, or or $x' = x_1 \diamond x_2$, with $\mathtt{v}\sigma x_1 \in \mathsf{GSS}(\alpha)$ and $x_2 \in \mathsf{GSS}(\beta)$. The former is equivalent to $\mathtt{v} \leq \mathsf{out}(\alpha)$, $\mathtt{v} \leq b$ and $x' \in \mathsf{GSS}(\gamma')$ for some $(b, \gamma') \in \partial_\sigma(\beta)$, i.e. to $\mathtt{v} \leq \mathsf{out}(\alpha)b$ and $x' \in \mathsf{GSS}(\gamma')$ for some $(\mathsf{out}(\alpha)b, \gamma') \in \partial_\sigma(\alpha\beta)$. The latter is equivalent to $\mathtt{v} \leq b$, $x_1 \in \mathsf{GSS}(\gamma')$ and $x_2 \in \mathsf{GSS}(\beta)$ for some $(b, \gamma') \in \partial_\sigma(\alpha)$, i.e. to $\mathtt{v} \leq b$ and $x' = x_1 \diamond x_2 \in \mathsf{GSS}(\gamma') \diamond \mathsf{GSS}(\beta) = \mathsf{GSS}(\gamma'\beta)$ for some $(b, \gamma'\beta) \in \partial_\sigma(\alpha\beta)$.

Consider $\gamma = \alpha \times \beta$ and $x = \mathtt{v}\sigma x'$. One has $x \in \mathsf{GSS}(\alpha \times \beta)$ iff $x \in \mathsf{GSS}(\alpha) \times \mathsf{GSS}(\beta)$. This means that either, $x = (\mathtt{v}\sigma_1 x_1) \times (\mathtt{v}\sigma_2 x_2)$ for some $\mathtt{v}\sigma_1 x_1 \in \mathsf{GSS}(\alpha)$, $\mathtt{v}\sigma_2 x_2 \in \mathsf{GSS}(\beta)$ such that $\sigma = \sigma_1 \cup \sigma_2$ and $x' = x_1 \times x_2$, or $\mathtt{v} \in \mathsf{GSS}(\alpha)$ and $x \in \mathsf{GSS}(\beta)$, or $\mathtt{v} \in \mathsf{GSS}(\beta)$ and $x \in \mathsf{GSS}(\alpha)$. The proof for the two last cases are analogous to the first case for the concatenation. On the other hand, $\mathtt{v}\sigma_1 x_1 \in \mathsf{GSS}(\alpha)$ and $\mathtt{v}\sigma_2 x_2 \in \mathsf{GSS}(\beta)$ is equivalent to $\mathtt{v} \leq b_1$, $x_1 \in \mathsf{GSS}(\gamma_1')$, $\mathtt{v} \leq b_2$ and $x_2 \in \mathsf{GSS}(\gamma_2')$ for some $(b_1, \gamma_1') \in \partial_{\sigma_1}(\alpha)$ and $(b_2, \gamma_2') \in \partial_{\sigma_2}(\beta)$, i.e. to $\mathtt{v} \leq b_1 \times b_2$, $x' = x_1 \times x_2 \in \mathsf{GSS}(\gamma_1') \times \mathsf{GSS}(\gamma_2') = \mathsf{GSS}(\gamma_1' \times \gamma_2')$ for some $(b_1 \times b_2, \gamma_1' \times \gamma_2') \in \partial_\sigma(\alpha \times \beta)$. ◻

Example 8. Consider the expressions $\alpha = (t_1 p)^\star \neg t_1$ and $\beta = (t_2 pq + \neg t_2 q)$, which represent the programs `while` t_1 `do` p and `if` t_2 `then` $p; q$ `else` q, respectively. The partial derivative automaton for $\alpha \times \beta$, corresponding to the synchronous execution of both programs is the following.



To test the equivalence of SKAT terms we can consider the algorithm that tests the equivalence of NTAs as presented by Broda *et al.* [10], and implicitly use the definition of the partial derivative automaton associated to an SKAT expression.

4 Experimental results

We implemented (in Python) the algorithm by Broda *et al.* for testing NTAs equivalence and performed some experiments [1]. The implementation uses BDDs (binary decision diagrams) for dealing with boolean functions. To compare the performance of the new NTAs with respect to the ones that use explicitly derivatives w.r.t $\mathtt{v}\sigma \in \mathsf{At}\Sigma$ we considered the same experiments as in Almeida [2, Section 3.5.2]. Each sample has 10000 KAT expressions generated uniformly at random of a given size. For each sample we performed two experiments: (1) we tested the equivalence of each KAT expression against itself; (2) we tested the equivalence of two consecutively generated KAT expressions. For each pair of KAT expressions we measured: the number of pairs of derivatives generated (H), the number of iterations (it), which gives an estimate of the boolean assignments that must be tested for each program symbol, and the number ($|\alpha|_{\mathsf{T}}$) of tests of T in each expression. Table 2 summarizes both the results obtained and the ones obtained by Almeida. Each row corresponds to a sample, where the three first columns characterize the sample, respectively, $|\mathsf{A_B}|$ (k), $|\mathsf{T}|$ (l), and the length of each KAT expression generated. Rows $a.$ to $e.$ contain our results, and corresponding ones obtained by Almeida are listed in rows $f.$ to $j.$ Column four has the number of elements of T in each expression ($|\alpha|_{\mathsf{T}}$). Columns five and seven give the average size of H in the experiment (1) and (2), respectively. Columns six and eight have the number of iterations. These two columns have no entries for Almeida's results, as in that algorithm all assignments of $|\mathsf{At}|$ are considered

[1] Source code at `http://www.dcc.fc.up.pt/~nam/web/resources/katexp.tgz`

for each symbol of $|\mathsf{A_B}|$. Finally, the last two columns are the average times, in seconds, of each experiment. For Almeida's results the implementation was in Ocaml and the values where obtained with an Intel® Xeon® 5140 at 2.33 GHz with 4 GB of RAM, whereas the new values were obtained with an AMD® Phenom(tm)® II X4 955 ar 3.20 GHz with 32 GB of RAM. The most significative cases are the two last ones, in *d.* and *e.* and in *i.* and *j.* respectively, where a substantial performance improvement was achieved with the new algorithm.

	1	2	3	4	5	6	7	8	9	10
	k	l	$\|\alpha\|$	$\|\alpha\|_{\mathsf{T}}$	$H(1)$	$it(1)$	$H(2)$	$it(2)$	Time(1)	Time(2)
a.	5	5	50	10.33	9.21	149	0.49	0.19	0.08552	0.00148
b.	5	5	100	19.55	15.74	2854	0.66	0.88	2.7256	0.00278
c.	10	10	50	10.32	11.61	59.56	0.30	0.03	0.05424	0.0035
d.	10	10	100	19.89	20.87	516	0.35	0.09	1.1969	0.01274
e.	15	15	50	10.31	12.78	50.7	0.25	0.013	0.0616	0.00738
f.	5	5	50	9.98	7.35	n.a.	0.53	n.a.	0.0097	0.00087
g.	5	5	100	19.71	15.74	n.a.	0.76	n.a.	0.0875	0.00223
h.	10	10	50	11.12	8.30	n.a.	0.50	n.a.	0.5050	0.30963
i.	10	10	100	21.93	16.78	n.a.	0.67	n.a.	20.45	1.31263
j.	15	15	50	11.57	8.47	n.a.	0.47	n.a.	6.4578	55.22

Table 2. Experimental results for uniformly random generated KAT expressions

Damien Pous developed an equivalence test for symbolic automata [31] and performed some tests for KAT terms [24]. To ensure equivalence of a pair of KAT terms (α_1, α_2) he added $\mathsf{A_B}^\star$ to each term. We ran a similar test considering a sample of 10000 pairs of KAT terms with $k = 7$, $l = 7$ and $|\alpha| = 100$. The values obtained were $|\alpha|_{\mathsf{T}} = 18.58$, $H = 41.34$, $it = 1745$ and $Time = 1.9456$, which are competitive with the ones in [24] (for Antimirov's algorithm).

5 Conclusion

In this paper we extended the notion of derivative to sets of (guarded) synchronous strings and showed that the methods based on derivatives lead to simple and elegant decision procedures for testing SKA and SKAT expressions equivalence. Based on our experiments, it may be worthwhile to study the average-case size of the SKA automata, in the analytic combinatorics framework. We also implemented the new class of SKAT automata based on NTAs automata. As the performance of testing NTA equivalence seems competitive we believe that our extension to SKAT automata is also much more efficient than the one proposed by Prisacariu.

References

1. Almeida, M., Moreira, N., Reis, R.: Testing regular languages equivalence. Journal of Automata, Languages and Combinatorics 15(1/2), 7–25 (2010)
2. Almeida, R.: Decision Algorithms for Kleene Algebra with Tests and Hoare Logic. Master's thesis, Faculdade de Ciências da Universidade do Porto (July 2012), `http://www.dcc.fc.up.pt/~nam/web/resources/docs/thesisRA.pdf`
3. Almeida, R., Broda, S., Moreira, N.: Deciding KAT and Hoare logic with derivatives. In: Faella, M., Murano, A. (eds.) 3rd GANDALF. EPTCS, vol. 96, pp. 127–140 (2012)
4. Antimirov, V.M.: Partial derivatives of regular expressions and finite automaton constructions. Theoret. Comput. Sci. 155(2), 291–319 (1996)
5. Berry, G., Gonthier, G.: The Esterel synchronous programming language: Design, semantics, implementation. Sci. Comput. Program. 19(2), 87–152 (1992)
6. Bonchi, F., Pous, D.: Checking NFA equivalence with bisimulations up to congruence. In: Giacobazzi, R., Cousot, R. (eds.) POPL '13. pp. 457–468. ACM (2013)
7. Braibant, T., Pous, D.: Deciding Kleene algebras in Coq. Logical Methods in Computer Science 8(1) (2012)
8. Broda, S., Machiavelo, A., Moreira, N., Reis, R.: On the average size of Glushkov and partial derivative automata. International Journal of Foundations of Computer Science 23(5), 969–984 (2012)
9. Broda, S., Machiavelo, A., Moreira, N., Reis, R.: On the average size of Glushkov and equation automata for KAT expressions. In: FCT 2013. pp. 72–83. No. 8070 in LNCS, Springer (2013)
10. Broda, S., Machiavelo, A., Moreira, N., Reis, R.: On the Equivalence of Automata for KAT-expressions. In: Beckmann, A., Csuhaj-Varjú, E., Meer, K. (eds.) CiE 2014. LNCS, vol. 8493, pp. 73–83. Springer (2014)
11. Broda, S., Machiavelo, A., Moreira, N., Reis, R.: Partial derivative automaton for regular expressions with shuffle. In: Okhotin, A., Shallit, J. (eds.) Proc. 17th DCFS 2015. LNCS, Springer (2015)
12. Coquand, T., Siles, V.: A decision procedure for regular expression equivalence in type theory. In: Jouannaud, J.P., Shao, Z. (eds.) CPP 2011, Kenting, Taiwan, December 7-9, 2011. pp. 119–134. No. 7086 in LNCS, Springer-Verlag (2011)
13. Glushkov, V.M.: The abstract theory of automata. Russian Math. Surveys 16, 1–53 (1961)
14. Hopcroft, J., Karp, R.M.: A linear algorithm for testing equivalence of finite automata. Tech. Rep. TR 71 -114, University of California, Berkeley, California (1971)
15. Kozen, D.: Kleene algebra with tests. Trans. on Prog. Lang. and Systems 19(3), 427–443 (05 1997)
16. Kozen, D.: On Hoare logic and Kleene algebra with tests. ACM Trans. Comput. Log. 1(1), 60–76 (2000)
17. Kozen, D.: Automata on guarded strings and applications. Matématica Contemporânea 24, 117–139 (2003)
18. Kozen, D.: On the coalgebraic theory of Kleene algebra with tests. Tech. Rep. `http://hdl.handle.net/1813/10173`, Cornell University (05 2008)
19. Krauss, A., Nipkow, T.: Proof pearl: Regular expression equivalence and relation algebra. Journal of Automated Reasoning (2011), published online
20. Milner, R.: Communication and concurrency. PHI Series in computer science, Prentice Hall (1989)

21. Moreira, N., Pereira, D., de Sousa, S.M.: Deciding regular expressions (in-)equivalence in Coq. In: Griffin, T.G., Kahl, W. (eds.) 13th RAMiCS 2012. LNCS, vol. 7560, pp. 98–113. Springer (2012)
22. Nipkow, T., Traytel, D.: Unified decision procedures for regular expression equivalence. Archive of Formal Proofs 2014 (2014)
23. Pereira, D.: Towards Certified Program Logics for the Verification of Imperative Programs. Ph.D. thesis, University of Porto (2013)
24. Pous, D.: Symbolic Algorithms for Language Equivalence and Kleene Algebra with Tests. In: Rajamani, S.K., Walker, D. (eds.) 42nd POPL 2015. pp. 357–368. ACM (2015)
25. Prisacariu, C.: Synchronous Kleene algebra. J. Log. Algebr. Program. 79(7), 608–635 (2010)
26. Project FAdo: FAdo: tools for formal languages manipulation. `http://fado.dcc.fc.up.pt/` (Access date:0142015)
27. Rot, J., Bonsangue, M.M., Rutten, J.J.M.M.: Coinductive proof techniques for language equivalence. In: Dediu, A.H., Martín-Vide, C., Truthe, B. (eds.) LATA 2013. LNCS, vol. 7810, pp. 480–492. Springer (2013)
28. Silva, A.: Position automata for Kleene algebra with tests. Sci. Ann. Comp. Sci. 22(2), 367–394 (2012)
29. Synopsys: Esterel studio. `http://www.synopsys.com/home.aspx`
30. Thompson, K.: Regular expression search algorithm. Communications of the ACM 11(6), 410–422 (1968)
31. Veanes, M.: Applications of symbolic finite automata. In: Konstantinidis, S. (ed.) Proc. 18th CIAA 2013. LNCS, vol. 7982, pp. 16–23. Springer (2013)