Automata for KAT Expressions

Sabine Broda António Machiavelo Nelma Moreira Rogério Reis
CMUP & DCC, Faculdade de Ciências da Universidade do Porto
Rua do Campo Alegre, 4169-007 Porto, Portugal

Technical Report Series: DCC-2014-01
Version 1.0 January 2014

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
Rua do Campo Alegre, 1021/1055,
4169-007 PORTO,
PORTUGAL
Tel: 220 402 900 Fax: 220 402 950
http://www.dcc.fc.up.pt/Pubs/

Automata for KAT Expressions

Sabine Broda António Machiavelo Nelma Moreira Rogério Reis
CMUP & DCC, Faculdade de Ciências da Universidade do Porto
Rua do Campo Alegre, 4169-007 Porto, Portugal

January 21, 2014

Abstract

Kleene algebra with tests (KAT) is a decidable equational system for program verification, that uses both Kleene and Boolean algebras. In spite of KAT 's elegance and success in providing theoretical solutions for several problems, not many efforts have been made towards obtaining tractable decision procedures that could be used in practical software verification tools. The main drawback of the existing methods relies on the explicit use of all possible assignments to boolean variables. Recently, Silva introduced an automata model that extends Glushkov's construction for regular expressions. Broda et al. extended also Mirkin's equation automata to KAT expressions and studied the state complexity of both algorithms. Contrary to other automata constructions from KAT expressions, these two constructions enjoy the same descriptional complexity behaviour as their counterparts for regular expressions, both in the worst case as well as in the average case. In this paper, we generalize, for these automata, the classical methods of subset construction for nondeterministic finite automata, and the Hopcroft and Karp algorithm for testing deterministic finite automata equivalence. As a result, we obtain a decision procedure for KAT equivalence where the extra burden of dealing with boolean expressions avoids the explicit use of all possible assignments to the boolean variables. Finally, we specialize the decision procedure for testing KAT expressions equivalence without explicitly constructing the automata, by introducing a new notion of derivative and a new method of constructing the equation automaton.

Keywords: Kleene algebra with tests, automata, equivalence, derivative.

1 Introduction

Kleene algebra with tests (KAT) [11] is an equational system that extends Kleene algebra (KA), the algebra of regular expressions, and that is specially suited to capture and verify properties of simple imperative programs. In particular, it subsumes the propositional Hoare logic which is a formal system for the specification and verification of programs, and that is, currently, the base of most of the tools for checking program correctness. The equational theory of KAT is PSPACE-complete and can be reduced to the equational theory of KA, with an exponential cost [8, 14]. Regular sets of guarded strings are standard models for KAT (as regular languages are for KA). The decidability, conciseness and expressiveness of KAT, motivated its recent automatization within several theorem provers [16, 17, 4] and functional languages [3]. Those implementations use (variants of) the coalgebraic automata on guarded strings developed by Kozen [13]. In this approach, derivatives are considered over symbols of the from αp, where p is an alphabetic symbol (program) and α a valuation of boolean variables (the *guard*, normally called atom). 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. Recently, Silva [18] introduced an automata model for KAT expressions that extends Glushkov's construction for regular expressions. In this automata, transitions are labeled with KAT expressions of the form bp, where b is a boolean expression (and not an atom) and p an alphabetic symbol. Using similar ideas, Broda et al. [6] extended the Mirkin's equation automata to KAT expressions and studied the state complexity of both algorithms.

Contrary to other automata constructions for KAT expressions, these two constructions enjoy the same descriptional complexity behaviour as their counterparts for regular expressions, both in the worst-case as well as in the average-case. In this paper, we generalize, for these automata, the classical methods of subset construction for nondeterministic finite automata, and the Hopcroft and Karp algorithm for testing deterministic finite automata equivalence. As a result, we obtain a decision procedure for KAT equivalence where the extra burden of dealing with boolean expressions avoids the explicit use of all possible assignments to the boolean variables. Finally, we specialize the decision procedure for testing KAT expressions equivalence without explicitly constructing the automata, by introducing a new notion of derivative and a new method of constructing the equation automaton.

2 KAT Expressions, Automata, and Guarded Strings

Let $\mathsf{P} = \{p_1, \ldots, p_k\}$ be a non-empty set, usual referred to as the set of *program* symbols, and $\mathsf{T} = \{t_0, \ldots, t_{l-1}\}$ be a non-empty set of *test* symbols. The set of boolean expressions over T together with negation, disjunction and conjunction, is denoted by BExp, and the set of KAT expressions with disjunction, concatenation, and Kleene star, by Exp. The abstract syntax of KAT expressions, over an alphabet $\mathsf{P} \cup \mathsf{T}$, is given by the following grammar, where $p \in \mathsf{P}$ and $t \in \mathsf{T}$,

$$\begin{array}{rrcl} \mathsf{BExp}: & b & \to & 0 \mid 1 \mid t \mid \neg b \mid b + b \mid b \cdot b \\ \mathsf{Exp}: & e & \to & p \mid b \mid e + e \mid e \cdot e \mid e^\star. \end{array}$$

As usual, we will omit the operator $\cdot$ whenever it does not give rise to any ambiguity. For the negation of test symbols we frequently use $\bar{t}$ instead of $\neg t$. The set At, of *atoms* over T, is the set of all boolean assignments to all elements of T, $\mathsf{At} = \{x_0 \cdots x_{l-1} \mid x_i \in \{t_i, \bar{t}_i\},\ t_i \in \mathsf{T}\}$. Each atom $\alpha \in \mathsf{At}$ has associated a binary word of l bits $(w_0 \cdots w_{l-1})$ where $w_i = 0$ if $\bar{t}_i \in \alpha$, and $w_i = 1$, if $t_i \in \alpha$.

Now, the set of *guarded strings* over P and T is $\mathsf{GS} = (\mathsf{At} \cdot \mathsf{P})^\star \cdot \mathsf{At}$. Regular sets of guarded strings form the standard language-theoretic model for KAT [12]. For $x = \alpha_1 p_1 \cdots p_{m-1}\alpha_m, y = \beta_1 q_1 \cdots q_{n-1}\beta_n \in \mathsf{GS}$, where $m, n \geq 1$, $\alpha_i, \beta_j \in \mathsf{At}$ and $p_i, q_j \in \mathsf{P}$, we define the *fusion product* $x \diamond y = \alpha_1 p_1 \cdots p_{m-1}\alpha_m q_1 \cdots q_{n-1}\beta_n$, if $\alpha_m = \beta_1$, leaving it undefined, otherwise. For sets $X, Y \subseteq \mathsf{GS}$, $X \diamond Y$ is the set of all $x \diamond y$ such that $x \in X$ and $y \in Y$. Let $X^0 = \mathsf{At}$ and $X^{n+1} = X \diamond X^n$, for $n \geq 0$. Given a KAT expression e, we define $\mathsf{GS}(e) \subseteq \mathsf{GS}$ inductively as follows:

$$\begin{array}{rcl} \mathsf{GS}(p) & = & \{\, \alpha p \beta \mid \alpha, \beta \in \mathsf{At} \,\} \\ \mathsf{GS}(b) & = & \{\, \alpha \mid \alpha \in \mathsf{At} \wedge \alpha \leq b \,\} \end{array} \qquad \begin{array}{rcl} \mathsf{GS}(e_1 + e_2) & = & \mathsf{GS}(e_1) \cup \mathsf{GS}(e_2) \\ \mathsf{GS}(e_1 \cdot e_2) & = & \mathsf{GS}(e_1) \diamond \mathsf{GS}(e_2) \\ \mathsf{GS}(e_1^\star) & = & \cup_{n \geq 0} \mathsf{GS}(e_1)^n, \end{array}$$

where $\alpha \leq b$ if $\alpha \to b$ is a propositional tautology. For $E \subseteq \mathsf{Exp}$, let $\mathsf{GS}(E) = \bigcup_{e \in E} \mathsf{GS}(e)$. Given two KAT expressions e_1 and e_2, we say that they are *equivalent*, and write $e_1 = e_2$, if $\mathsf{GS}(e_1) = \mathsf{GS}(e_2)$.

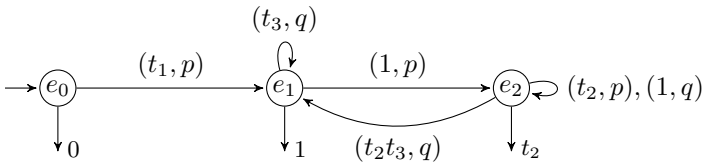
A *(nondeterministic) automaton with tests* (NTA) over the alphabets P and T is a tuple $\mathcal{A} = \langle S, s_0, o, \delta \rangle$, where S is a finite set of states, $s_0 \in S$ is the initial state, $o : S \to \mathsf{BExp}$ is the output function, and $\delta \subseteq 2^{S \times (\mathsf{BExp} \times \mathsf{P}) \times S}$ is the transition relation. We denote by $\mathsf{P}_\mathcal{A}$ and $\mathsf{BExp}_\mathcal{A}$, respectively, the set of program symbols and the set of boolean expressions that occur in δ. In general, we assume that there are no transitions $(s, (b, p), s') \in \delta$ such that b is not satisfiable.

A guarded string $\alpha_1 p_1 \ldots p_{n-1}\alpha_n$, with $n \geq 1$, is accepted by the automaton $\mathcal{A}$ if and only if there is a sequence of states $s_0, s_1, \ldots, s_{n-1} \in S$, where s_0 is the initial state, and, for $i = 1, \ldots, n-1$, one has $\alpha_i \leq b_i$ for some $(s_{i-1}, (b_i, p_i), s_i) \in \delta$, and $\alpha_n \leq o(s_{n-1})$. The set of all guarded strings accepted by $\mathcal{A}$ is denoted by $\mathsf{GS}(\mathcal{A})$. Formally, given an NTA $\mathcal{A} = \langle S, s_0, o, \delta \rangle$, one can naturally associate to the transition relation $\delta \subseteq 2^{S \times (\mathsf{BExp} \times \mathsf{P}) \times S}$ a function $\delta' : S \times (\mathsf{At} \cdot \mathsf{P}) \longrightarrow 2^S$, defined by $\delta'(s, \alpha p) = \{\, s' \mid (s, (b, p), s') \in \delta,\ \alpha \leq b \,\}$. And, one can define a function $\hat{\delta} : S \times \mathsf{GS} \longrightarrow \{0, 1\}$ over pairs of states and guarded strings as follows

$$\hat{\delta}(s, \alpha) = \begin{cases} 1 & \text{if } \alpha \leq o(s), \\ 0 & \text{otherwise,} \end{cases} \qquad \hat{\delta}(s, \alpha p x) = \sum_{s' \in \delta'(s, \alpha p)} \hat{\delta}(s', x) \ .$$

Given a state s, $\mathsf{GS}(s) = \{\ x \in \mathsf{GS} \mid \hat{\delta}(s,x) = 1\ \}$ is the set of guarded strings accepted by s, and $\mathsf{GS}(\mathcal{A}) = \mathsf{GS}(s_0)$. We say that a KAT expression $e \in \mathsf{Exp}$ is *equivalent* to an automaton $\mathcal{A}$, and write $e = \mathcal{A}$, if $\mathsf{GS}(\mathcal{A}) = \mathsf{GS}(e)$.

Example 1. *Given the KAT expression $e = t_1 p(pq^\star t_2 + t_3 q)^\star$, an equivalent NTA $\mathcal{A}$, obtained by the equation algorithm (see [6]), is the following, where $e_0 = e$, $e_1 = (pq^\star t_2 + t_3 q)^\star$ and $e_2 = q^\star t_2 (pq^\star t_2 + t_3 q)^\star$. Both objects accept the guarded string $t_1 t_2 t_3 p t_1 t_2 t_3 p t_1 t_2 t_3 q t_1 t_2 t_3$.*



An NTA is called *deterministic* (DTA) if and only if for every pair $(\alpha, p) \in \mathsf{At} \times \mathsf{P}$ and every state $s \in S$, there is at most one transition $(s, (b,p), s') \in \delta$ such that $\alpha \leq b$, i.e. $\delta'(s, \alpha p)$ is either empty or a singleton.

3 Determinization

The standard subset construction for converting a nondeterministic finite automaton (NFA) into an equivalent deterministic finite automaton (DFA) may be adapted as follows. Given a set of states $X \subseteq S$, whenever there are transitions $(s_1, (b_1, p), s_1'), \ldots, (s_m, (b_m, p), s_m')$, with $s_1, \ldots, s_m \in X$, in the NTA, in the equivalent DTA we consider "disjoint" transitions to subsets $\{s_{i_1}', \ldots, s_{i_k}'\} \subseteq \{s_1', \ldots, s_m'\}$, labeled by $(b_{i_1} \cdots b_{i_k} \neg b_{i_{k+1}} \cdots \neg b_{i_m})p$, where $\{s_{i_{k+1}}', \ldots, s_{i_m}'\} = \{s_1', \ldots, s_m'\} \setminus \{s_{i_1}', \ldots, s_{i_k}'\}$.

Consider the set of atoms $\mathsf{At} = \{\alpha_0, \ldots, \alpha_{2^l - 1}\}$, with the natural order induced by their binary representation. We define the function

$$\begin{array}{rccl} \mathsf{V}: & \mathsf{BExp} & \longrightarrow & 2^{\{0,\ldots,2^l-1\}} \\ & b & \longmapsto & \mathsf{V}_b = \{\ i \mid \alpha_i \leq b, 0 \leq i \leq 2^l - 1\ \}. \end{array}$$

This representation of boolean expressions is such that $\mathsf{V}_b = \mathsf{V}_{b'}$ if and only if b and b' are logically equivalent expressions. We consider V_b as a canonical representation of b and write $\alpha_i \leq \mathsf{V}_b$ if and only if $i \in \mathsf{V}_b$. Conversely, to each $U \subseteq \{0, \ldots, 2^l - 1\}$ we associate a unique boolean expression $\mathsf{B}(U)$, where

$$\begin{array}{rccl} \mathsf{B}: & 2^{\{0,\ldots,2^l-1\}} & \longrightarrow & \mathsf{BExp} \\ & U & \longmapsto & \displaystyle\sum_{i \in U} \alpha_i. \end{array}$$

For $b, b' \in \mathsf{BExp}$ we have $\mathsf{V}_{\neg b} = \overline{\mathsf{V}}_b$, $\mathsf{V}_{b+b'} = \mathsf{V}_b \cup \mathsf{V}_{b'}$ and $\mathsf{V}_{b \cdot b'} = \mathsf{V}_b \cap \mathsf{V}_{b'}$, where $\overline{U} = \{0, \ldots, 2^l - 1\} \setminus U$ for any $U \subseteq \{0, \ldots, 2^l - 1\}$.

Example 2. *For $\mathsf{T} = \{t_1, t_2\}$ and $\mathsf{At} = \{\bar{t}_1\bar{t}_2, \bar{t}_1 t_2, t_1\bar{t}_2, t_1 t_2\}$, we have $\mathsf{V}_{t_2} = \{1, 3\}$ and $\mathsf{V}_{t_1 + \neg t_2} = \{0, 1, 3\}$. Also, $\mathsf{B}(\{1,3\}) = t_1 t_2 + \bar{t}_1 t_2$.*

We now describe the subset construction that, given an NTA, $\mathcal{A} = \langle S, s_0, o, \delta \rangle$, over the alphabets P and T, produces an DTA, $\mathcal{A}_{\mathsf{det}} = \langle 2^S, \{s_0\}, o_{\mathsf{det}}, \delta_{\mathsf{det}} \rangle$, over P and T, such that $\mathsf{GS}(\mathcal{A}) = \mathsf{GS}(\mathcal{A}_{\mathsf{det}})$. First, we define two functions

$$\tilde{\delta}_{\mathsf{det}} : 2^S \times (2^{\{0,\ldots,2^l-1\}} \times \mathsf{P}) \longrightarrow 2^S \quad \text{and} \quad \tilde{o}_{\mathsf{det}} : 2^S \longrightarrow 2^{\{0,\ldots,2^l-1\}}.$$

Then, we take $\delta_{\mathsf{det}} = \{\ (X, (\mathsf{B}(V), p), Y) \mid (X, (V, p), Y) \in \tilde{\delta}_{\mathsf{det}}\ \}$ as well as $o_{\mathsf{det}} = \mathsf{B} \circ \tilde{o}_{\mathsf{det}}$. For $X \subseteq S$, we define $\tilde{o}_{\mathsf{det}}(X) = \bigcup_{s \in X} \mathsf{V}_{o(s)}$. To define $\tilde{\delta}_{\mathsf{det}}$, we consider the following sets. Given $X \subseteq S$ and $p \in \mathsf{P}$, let

$$\begin{aligned} \Gamma(X, p) &= \{\ (b, s') \mid (s, (b, p), s') \in \delta,\ s \in X\ \}, \\ \Delta(X, p) &= \{\ s' \mid (b, s') \in \Gamma(X, p)\ \}. \end{aligned}$$

For $s' \in \Delta(X,p)$, we define $\mathsf{V}_{X,p,s'} = \bigcup \{ \mathsf{V}_b \mid (b,s') \in \Gamma(X,p) \}$, and for each $Y \subseteq \Delta(X,p)$ the set

$$\mathsf{V}_{X,p,Y} = \bigcap \left(\{ \mathsf{V}_{X,p,s'} \mid s' \in Y \} \cup \{ \overline{\mathsf{V}}_{X,p,s'} \mid s' \in \Delta(X,p) \setminus Y \} \right).$$

Finally, we have

$$\tilde{\delta}_{\mathsf{det}} = \{ (X, (\mathsf{V}_{X,p,Y}, p), Y) \mid X \subseteq S,\ Y \subseteq \Delta(X,p),\ p \in \mathsf{P},\ \mathsf{V}_{X,p,Y} \neq \emptyset \}.$$

Proposition 3. *For every NTA $\mathcal{A} = \langle S, s_0, o, \delta \rangle$, the automaton $\mathcal{A}_{\mathsf{det}} = \langle 2^S, \{s_0\}, o_{\mathsf{det}}, \delta_{\mathsf{det}} \rangle$ is deterministic.*

Proof. For all $X, Y \subseteq S$ and $p \in \mathsf{P}$, the set $\mathsf{V}_{X,p,Y}$ is uniquely defined (and equal to $\emptyset$, if $X = \emptyset$ or $Y = \emptyset$). Thus, it remains to show that whenever $(X, (\mathsf{V}_{X,p,Y}, p), Y), (X, (\mathsf{V}_{X,p,Y'}, p), Y') \in \tilde{\delta}_{\mathsf{det}}$, for some non-empty sets $X, Y, Y' \subseteq S$, $Y \neq Y'$ and $p \in \mathsf{P}$, then $\mathsf{V}_{X,p,Y} \cap \mathsf{V}_{X,p,Y'} = \emptyset$. Since $Y \neq Y'$, there is some $s' \in S$ such that $s' \in Y$ and $s' \notin Y'$ (or vice-versa). By the definition of $\mathsf{V}_{X,p,Y}$, we conclude that $\mathsf{V}_{X,p,Y} \subseteq \mathsf{V}_{X,p,s'}$ and $\mathsf{V}_{X,p,Y'} \subseteq \overline{\mathsf{V}}_{X,p,s'}$. Thus, $\mathsf{V}_{X,p,Y} \cap \mathsf{V}_{X,p,Y'} = \emptyset$. □

Corollary 4. *For every NTA $\mathcal{A} = \langle S, s_0, o, \delta \rangle$, $X \subseteq S$, $\alpha \in \mathsf{At}$ and $p \in \mathsf{P}$, the set $\delta'_{\mathsf{det}}(X, \alpha p)$ is either $\emptyset$ or a singleton $\{Y\}$, where $Y = \{ s' \mid (b,s') \in \Gamma(X,p),\ \alpha \leq b \}$.*

Proof. By definition, $\delta'_{\mathsf{det}}(X, \alpha p) = \{ Y \mid (X, (\mathsf{V}_{X,p,Y}, p), Y) \in \tilde{\delta}_{\mathsf{det}},\ \alpha \leq \mathsf{V}_{X,p,Y} \}$ and has to be either $\emptyset$ or a singleton since $\mathcal{A}_{\mathsf{det}}$ is deterministic. Furthermore, it follows, by the construction of $\mathsf{V}_{X,p,Y}$, that one has $(X, (\mathsf{V}_{X,p,Y}, p), Y) \in \tilde{\delta}_{\mathsf{det}}$ and $\alpha \leq \mathsf{V}_{X,p,Y}$ if and only if Y is exactly the set of states s' such that for some $b \in \mathsf{BExp}$ there is $(b,s') \in \Gamma(X,p)$ with $\alpha \leq b$. □

Proposition 5. *For $\mathcal{A} = \langle S, s_0, o, \delta \rangle$ one has $\mathsf{GS}(\mathcal{A}) = \mathsf{GS}(\mathcal{A}_{\mathsf{det}})$.*

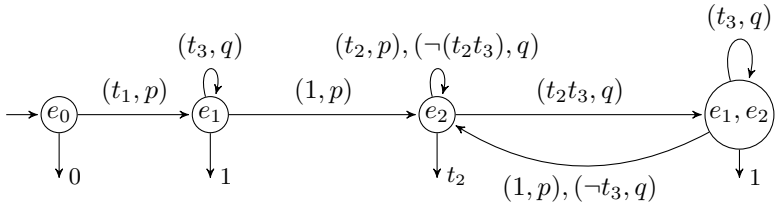
Proof. We will prove that for every set of states $X \subseteq S$ and every guarded string $x \in \mathsf{GS}$, one has $\hat{\delta}_{\mathsf{det}}(X, x) = \sum_{s \in X} \hat{\delta}(s, x)$. Thus, in particular $\hat{\delta}_{\mathsf{det}}(\{s_0\}, x) = \hat{\delta}(s_0, x)$. The proof is by induction on the number of action symbols in x.

For $x = \alpha$ we have $\sum_{s \in X} \hat{\delta}(s, \alpha) = 1$ if and only if there is some $s \in X$ such that $\alpha \leq o(s)$, i.e. $\alpha \leq \mathsf{V}_{o(s)}$. This is equivalent to $\alpha \leq o_{\mathsf{det}}(X)$, hence to $\hat{\delta}_{\mathsf{det}}(X, \alpha) = 1$. For $x = \alpha p y$, we have

$$\sum_{s \in X} \hat{\delta}(s, \alpha p y) = \sum_{\substack{s \in X \\ s' \in \delta'(s, \alpha p)}} \hat{\delta}(s', y) = \sum_{\substack{s \in X \\ (s,(b,p),s') \in \delta \\ \alpha \leq b}} \hat{\delta}(s', y) = \sum_{s' \in Y} \hat{\delta}(s', y),$$

where $Y = \{ s' \mid (b,s') \in \Gamma(X,p),\ \alpha \leq b \}$. By the induction hypothesis, we conclude that $\sum_{s \in X} \hat{\delta}(s, \alpha p y) = \sum_{s' \in Y} \hat{\delta}(s', y) = \hat{\delta}_{\mathsf{det}}(Y, y)$. On the other hand, it follows from Corollary 4 that $\hat{\delta}_{\mathsf{det}}(X, \alpha p y) = \hat{\delta}_{\mathsf{det}}(Y, y)$, with $Y = \{ s' \mid (b,s') \in \Gamma(X,p),\ \alpha \leq b \}$. □

Example 6. *Applying the construction above to the NTA from Example 1, we obtain the following DTA:*



To illustrate the construction, we first consider the subset $X = \{e_2\}$ and program symbol q. We have

$$\Gamma(\{e_2\}, q) = \{(t_2 t_3, e_1), (1, e_2)\} \quad \textit{and} \quad \Delta(\{e_2\}, q) = \{e_1, e_2\}.$$

Remember that for $\mathsf{T} = \{t_1, t_2, t_3\}$ *each element of* $\mathsf{At} = \{t_1t_2t_3, \ldots, \bar{t}_1\bar{t}_2\bar{t}_3\}$ *is represented by some* $i \in \{0, \ldots, 7\}$. *Furthermore,*

$$\mathsf{V}_{\{e_2\},q,e_1} = \{0, 4\} \quad \textit{and} \quad \mathsf{V}_{\{e_2\},q,e_2} = \{0, \ldots, 7\}.$$

Hence, we conclude that

$$\begin{array}{rlcl} & \mathsf{V}_{\{e_2\},q,\{e_1,e_2\}} & = & \bigcap \{\{0,4\},\{0,\ldots,7\}\} = \{0,4\} = \mathsf{V}_{t_2t_3} \\ & \mathsf{V}_{\{e_2\},q,\{e_1\}} & = & \bigcap \{\{0,4\},\{\}\} = \{\} \\ \textit{and} & \mathsf{V}_{\{e_2\},q,\{e_2\}} & = & \bigcap \{\{0,\ldots,7\},\{1,2,3,5,6,7\}\} = \{1,2,3,5,6,7\} = \mathsf{V}_{\neg(t_2t_3)}. \end{array}$$

Regarding $X = \{e_1, e_2\}$, *we have*

$$\Gamma(\{e_1, e_2\}, q) = \{(t_3, e_1), (t_2t_3, e_1), (1, e_2)\} \quad \textit{and} \quad \Delta(\{e_1, e_2\}, q) = \{e_1, e_2\}.$$

Furthermore,

$$\mathsf{V}_{\{e_1,e_2\},q,e_1} = \{0, 2, 4, 6\} \quad \textit{and} \quad \mathsf{V}_{\{e_1,e_2\},q,e_2} = \{0, \ldots, 7\}.$$

Finally,

$$\begin{array}{rlcl} & \mathsf{V}_{\{e_1,e_2\},q,\{e_1,e_2\}} & = & \bigcap \{\{0,2,4,6\},\{0,\ldots,7\}\} = \{0,2,4,6\} = \mathsf{V}_{t_3} \\ & \mathsf{V}_{\{e_1,e_2\},q,\{e_1\}} & = & \bigcap \{\{0,2,4,6\},\{\}\} = \{\} \\ \textit{and} & \mathsf{V}_{\{e_1,e_2\},q,\{e_2\}} & = & \bigcap \{\{0,\ldots,7\},\{1,3,5,7\}\} = \{1,3,5,7\} = \mathsf{V}_{\neg t_3}. \end{array}$$

3.1 Implementation and Complexity

It is important to notice that in the determinization algorithm, the construction of all the $2^{|S|}$ subsets X of the set of states S can be avoided by considering only reachable states from the initial state. In order to, efficiently deal with boolean operations it is essential to have an adequate representation for the boolean expressions b as well as the sets V_b. A possible choice is to use OBDDs (ordered binary decision diagrams), for which there are several software packages available. The sets $\mathsf{V}_{X,p,Y}$ may also be constructed using a (variant) of the standard Quine-McCluskey algorithm.

In the worst case, the determinization algorithm exhibits an extra exponential complexity to compute the sets $\mathsf{V}_{X,p,Y}$. The deterministic automaton $\mathcal{A}_{\mathsf{det}}$ has at most 2^n states and $2^n 2^l k$ transitions where $n = |S|$, $l = |\mathsf{T}|$, and $k = |\mathsf{P}|$. Contrary to what happens with the other KAT automata where the set At is explicitly used, in practice and with adequate data structures, we can expect that the number of sets $X \subseteq S$ and of sets $\mathsf{V}_{X,p,Y}$ is kept within tractable limits. It is an open problem to theoretically obtain the average-case complexity of both the power set construction and the sets $\mathsf{V}_{X,p,Y}$.

4 Equivalence of Deterministic Automata

Hopcroft and Karp [10] presented an almost linear algorithm (HK) for testing the equivalence of two DFAs that avoids their minimization. Considering the merge of the two DFAs as a single one, the algorithm computes the finest right-invariant relation, on the set of states, that makes the initial states equivalent. Recently this algorithm was analyzed and extended to NFAs in [2, 5]. In this section, we extend it, again, for testing equivalence of deterministic automata for guarded strings. We will only consider DTAs, $\mathcal{A} = \langle S, s_0, o, \delta\rangle$, where all states are useful, i.e. for every state $s \in S$, $\mathsf{GS}(s) \neq \emptyset$.

Given a DTA, $\mathcal{A} = \langle S, s_0, o, \delta\rangle$, over the alphabets P and T, and $s, t \in S$, we say that s and t are *equivalent*, and write $s \approx t$, if $\mathsf{GS}(s) = \mathsf{GS}(t)$. A binary relation R on S is *right invariant* if for all $s, t \in S$ if sRt then the following conditions hold:

- $\forall \alpha \in \mathsf{At},\ \alpha \leq o(s) \Leftrightarrow \alpha \leq o(t)$;
- $\forall \alpha p \in \mathsf{At} \cdot \mathsf{P},\ (\delta'(s, \alpha p) = \delta'(t, \alpha p) = \emptyset)$ or $(\delta'(s, \alpha p) = \{s'\},\ \delta'(t, \alpha p) = \{t'\}$ and $s'Rt')$.

It is easy to see that the relation $\approx$ is right invariant. Furthermore, whenever R is a right-invariant relation on S and sRt, for $s, t \in S$, one has $s \approx t$.

Let $\mathcal{A}_1 = \langle S_1, s_0, o_1, \delta_1 \rangle$ and $\mathcal{A}_2 = \langle S_2, r_0, o_2, \delta_2 \rangle$ be two DTAs over the alphabets P and T, such that $S_1 \cap S_2 = \emptyset$. The algorithm HK, given below, decides if these two automata are equivalent, i.e. if $\mathsf{GS}(\mathcal{A}_1) = \mathsf{GS}(\mathcal{A}_2)$, by building a right-invariant relation that checks whether $s_0 \approx r_0$. Consider $\mathcal{A} = \langle S, s_0, o, \delta \rangle$, where

$$S = S_1 \cup S_2, \qquad o(s) = \begin{cases} o_1(s) & \text{if } s \in S_1 \\ o_2(s) & \text{if } s \in S_2 \end{cases} \qquad \text{and} \quad \delta = \delta_1 \cup \delta_2.$$

Lemma 7. *Given two DTAs, $\mathcal{A}_1$ and $\mathcal{A}_2$, let $\mathcal{A}$ be defined as above. Then, $s_0 \approx r_0$ (in $\mathcal{A}$) if and only if $\mathsf{GS}(\mathcal{A}_1) = \mathsf{GS}(\mathcal{A}_2)$.*

The algorithm uses an initially empty stack H and a set partition $\mathcal{P}$ of S, which are both updated during the computation. The set partition $\mathcal{P}$ is built using the UNION-FIND data structure [9]. Within this structure, three functions are defined:

- MAKE(i): creates a new set (singleton) for one element i (the identifier);
- FIND(i): returns the identifier S_i of the set which contains i;
- UNION(i, j, k): combines the sets identified by i and j into a new set $S_k = S_i \cup S_j$; S_i and S_j are destroyed.

An arbitrary sequence of i operations MAKE, UNION, and FIND, j of which are MAKE operations, necessary to create the required sets can, in worst-case, be performed $O(i\alpha(j))$ time, where $\alpha(j)$ is related to a functional inverse of the Ackermann function, and, as such, grows very slowly, and for practical uses can be considered a constant. In the whole we assume that whenever FIND(i) fails, MAKE(i) is called.

Algorithm 1: HK algorithm for deterministic automata.

```
1  def HK(A1, A2):
2    MAKE(s0); MAKE(r0)
3    H = ∅
4    UNION(s0, r0, r0); PUSH(H, (s0, r0))
5    while (s, t) = POP(H):
6      if V_{o(s)} ≠ V_{o(t)}: return False
7      for p ∈ P:
8        B1 = Γ({s}, p)
9        B2 = Γ({t}, p)
10       if ⋃_{(b1,_)∈B1} V_{b1} = ⋃_{(b2,_)∈B2} V_{b2}:
11         for (b1, s') ∈ B1:
12           for (b2, t') ∈ B2:
13             if V_{b1} ∩ V_{b2} ≠ ∅:
14               s' = FIND(s')
15               t' = FIND(t')
16               if s' ≠ t':
17                 UNION(s', t', t')
18                 PUSH(H, (s', t'))
19       else: return False
20   return True
```

The algorithm terminates because every time it pushes a pair onto the stack it performs a union of two disjoint sets in the partition, and this can be done at most $|S| - 1$ times. Given that set operations introduce what can be considered a constant time factor, the worst-case running time of the algorithm HK is $O(m^2kn)$, where $n = |S|$, $k = |\mathsf{P}_{\mathcal{A}}|$, and $m = |\mathsf{BExp}_{\mathcal{A}}|$. The correctness of this version of algorithm HK is given by the proposition below, whose proof follows closely the one for DFAs.

Definition 8. *We define a binary relation* $\sim_{\mathsf{HK}} \subseteq S \times S$ *by* $s \sim_{\mathsf{HK}} t$ *if FIND(s) = FIND(t), when the algorithm terminates and returns* True.

This relation is obviously an equivalence relation.

Lemma 9. *The relation* $\sim_{\mathsf{HK}}$ *is right invariant.*

Proof. If $s \sim_{\mathsf{HK}} t$ and $s \neq t$, then there exists a sequence of states $s = q_1, q_2, \ldots, q_n = t \in FIND(s)$, with $n \geq 2$, such that either $(q_i, q_{i+1}) \in H$ or $(q_{i+1}, q_i) \in H$ at some point of the computation, for $i = 1, \ldots, n-1$. Since $\mathsf{V}_{o(q_i)} = \mathsf{V}_{o(q_{i+1})}$ (otherwise the algorithm would have failed in line 6), we have $\forall \alpha \in \mathsf{At} \quad \alpha \leq o(q_i) \Leftrightarrow \alpha \leq o(q_{i+1})$. Furthermore, it follows from the test in line 10 and from the automaton being deterministic, that $\forall \alpha p \in \mathsf{At} \cdot \mathsf{P}$ either $\delta'(q_i, \alpha p) = \delta'(q_{i+1}, \alpha p) = \emptyset$ or $\delta'(q_i, \alpha p) = \{s'\}$, $\delta'(q_{i+1}, \alpha p) = \{t'\}$. In this last case, there exist $(q_i, (b_1, p), s'), (q_{i+1}, (b_2, p), t') \in \delta$ such that $\alpha \leq b_1 \cdot b_2$. Consequently, either one has already $FIND(q_i) = FIND(q_{i+1})$, or (s', t') enters H and one has $FIND(q_i) = FIND(q_{i+1})$ afterwards. Thus, $s' \sim_{\mathsf{HK}} t'$. Since these properties are true for all $i = 1, \ldots, n-1$, we conclude that $\forall \alpha \in \mathsf{At} \quad \alpha \leq o(s) \Leftrightarrow \alpha \leq o(t)$. Furthermore, $\forall \alpha p \in \mathsf{At} \cdot \mathsf{P}$ either $\delta'(s, \alpha p) = \delta'(t, \alpha p) = \emptyset$ or $\delta'(s, \alpha p) = \{s'\}$, $\delta'(t, \alpha p) = \{t'\}$ and $s \sim_{\mathsf{HK}} t$. □

Proposition 10. *The algorithm returns* True *if and only if* $s_0 \approx r_0$.

Proof. If the algorithm returns True, then $s_0 \sim_{\mathsf{HK}} r_0$. Since $\sim_{\mathsf{HK}}$ is right invariant, we conclude that $s_0 \approx r_0$.

If $s_0 \not\approx r_0$, then there is some $x = \alpha_1 p_1 \ldots \alpha_n p_n \alpha \in \mathsf{GS}$, such that $\hat{\delta}(s_0, x) = 1$ and $\hat{\delta}(r_0, x) = 0$ (or vice-versa). $\hat{\delta}(s_0, x) = 1$ implies that there are $s_1, \ldots, s_n \in S$ such that $\alpha \leq o(s_n)$ and for $i = 1, \ldots, n$, there is $(s_{i-1}, (b_i, p_i), s_i) \in \delta$ with $\alpha_i \leq b_i$. One reason for $\hat{\delta}(r_0, x) = 0$ is the existence of $k \in \{1, \ldots, n-1\}$ such that there are $t_1, \ldots, t_k \in S$ such that for $i = 1, \ldots, k$, there is $(t_{i-1}, (b'_i, p_i), t'_i) \in \delta$ with $\alpha_i \leq b'_i$, but there is no $(t_k, (b'_{k+1}, p_{k+1}), t'_{k+1}) \in \delta$ with $\alpha_{k+1} \leq b'_{k+1}$. In this case, it is easy to verify, by induction on k, that one has $FIND(s_1) = FIND(t_1), \ldots, FIND(s_k) = FIND(t_k)$, but the algorithm will fail in step 10 after constructing B_1 and B_2 respectively for $s = FIND(s_k)$ and $t = FIND(t_k)$ (this, in case it has not already failed before). The other possible reason is that there are actually $t_1, \ldots, t_n \in S$ such that for $i = 1, \ldots, n$, there is $(t_{i-1}, (b'_i, p_i), t_i) \in \delta$ with $\alpha_i \leq b'_i$, but $\alpha \not\leq o(t_n)$. In this case, the algorithm fails at step 6. □

5 Equivalence of Nondeterministic Automata

We can embed the determinization process directly into the HK algorithm, extending it, so that it can be used to test the equivalence of NTAs. As before, given two NTAs with disjoint sets of states, $\mathcal{A}_1$ and $\mathcal{A}_2$, we consider them as a single NTA, $\mathcal{A} = \langle S, s_0, o, \delta \rangle$. We denote the resulting algorithm by HKN.

Algorithm 2: HKN algorithm for nondeterministic automata.

```
1  def HKN(A_1, A_2):
2    MAKE({s_0}); MAKE({r_0})
3    H = ∅
4    UNION({s_0},{r_0},{r_0}); PUSH(H,({s_0},{r_0}))
5    while (X,Y) = POP(H):
6      if õ_det(X) ≠ õ_det(Y): return False
7      for p ∈ P:
8        B_1 = Γ(X,p)
9        B_2 = Γ(Y,p)
10       if ⋃_{(b_1,-)∈B_1} V_{b_1} = ⋃_{(b_2,-)∈B_2} V_{b_2}:
11         for X' ⊆ Δ(X,p):
12           for Y' ⊆ Δ(Y,p):
13             if V_{X,p,X'} ∩ V_{Y,p,Y'} ≠ ∅:
14               X' = FIND(X')
15               Y' = FIND(Y')
```

```
16                    if X' ≠ Y' :
17                      UNION(X', Y', Y')
18                      PUSH(H, (X', Y'))
19          else: return False
20      return True
```

6 Equivalence of KAT Expressions

Given two KAT expressions, e_1 and e_2, their equivalence can be tested by first converting each expression to an equivalent NTA and then, either by determinizing both and applying the HK algorithm (Section 4), or by directly using the resulting NTAs in algorithm HKN (Section 5). In particular, we could use the equation construction given in [6] to obtain NTAs equivalent to the given KAT expressions and then apply the HKN algorithm. The equation automata for KAT expressions is an adaptation of Mirkin's construction [15] for regular expressions. Given $e_0 \equiv e \in \mathsf{Exp}$, a set of KAT expressions $\pi(e) = \{e_1, \ldots, e_n\}$ is defined inductively by $\pi(p) = \{1\}$, $\pi(b) = \emptyset$, $\pi(e+f) = \pi(e) \cup \pi(f)$, $\pi(e \cdot f) = \pi(e)f \cup \pi(f)$, and $\pi(e^\star) = \pi(e)e^\star$.. This set satisfy the following system of equations

$$e_i = \sum_{j=1}^{n} b_{ij1} p_1 e_j + \cdots + \sum_{j=1}^{n} b_{ijk} p_k e_j + \mathsf{out}(e_i) \tag{1}$$

for $i = 0, \ldots, n$, $p_r \in \mathsf{P}$, $k = |\mathsf{P}|$, some $b_{ijr} \in \mathsf{BExp}$, and where function out is defined below. The equation automaton is $\mathcal{A}_{\mathsf{eq}}(e) = \langle \{e\} \cup \pi(e), e, \mathsf{out}, \delta_{\mathsf{eq}} \rangle$ with $\delta_{\mathsf{eq}} = \{\ (e_i, (b_{ijr}, p_r), e_j) \mid$ if $b_{ijr} p_r e_j$ is a component of the equation for e_i $\}$.

In this section, we will use this construction to define an algorithm for testing equivalence of KAT expressions by a recursive computation of their derivatives without explicitly building any automaton. However, the correctness of this procedure is justified by the correctness of the equation automaton. We first present a slightly different formalization of this automaton construction, which is more adequate for our purposes. The construction resembles the partial derivative automaton for regular expressions (that is known to be identical to the Mirkin automaton [7]). The resulting decision procedure for KAT equivalence is also similar to the ones recently presented for regular expressions (see [1, 2]) and can be seen as a *syntactic* (and more *compact*) version of the one presented by Kozen [13].

For $e \in \mathsf{Exp}$ and a program symbol $p \in \mathsf{P}$, the set $\partial_p(e)$ of *partial derivatives* of e w.r.t. p is inductively defined as follows:

$$\partial_p : \mathsf{Exp} \longrightarrow \mathsf{BExp} \times \mathsf{Exp}$$

$$\partial_p(p') = \begin{cases} \{(1,1)\} & \text{if } p' \equiv p \\ \emptyset & \text{otherwise} \end{cases} \qquad \partial_p(b) = \emptyset$$

$$\begin{aligned} \partial_p(e+e') &= \partial_p(e) \cup \partial_p(e') \\ \partial_p(ee') &= \partial_p(e)e' \cup \mathsf{out}(e)\partial_p(e') \\ \partial_p(e^\star) &= \partial_p(e)e^\star \end{aligned}$$

where $\mathsf{out} : \mathsf{Exp} \longrightarrow \mathsf{BExp}$ is defined by

$$\mathsf{out}(p) = 0 \qquad \mathsf{out}(b) = b$$

$$\begin{aligned} \mathsf{out}(e_1 + e_2) &= \mathsf{out}(e_1) + \mathsf{out}(e_2) \\ \mathsf{out}(e_1 \cdot e_2) &= \mathsf{out}(e_1) \cdot \mathsf{out}(e_2) \end{aligned} \qquad \mathsf{out}(e^\star) = 1.$$

and for $R \subseteq \mathsf{BExp} \times \mathsf{Exp}$, $e \in \mathsf{Exp}$, and $b \in \mathsf{BExp}$, $Re = \{\ (b', e'e) \mid (b', e') \in R\ \}$ and $bR = \{\ (bb', e') \mid (b', e') \in R\ \}$. We also define $\Delta_p(e) = \{\ e' \mid (b, e') \in \partial_p(e)\ \}$. The functions ∂_p, out, and Δ_p are naturally extended to sets $X \subseteq \mathsf{Exp}$. Moreover we define the KAT expression $\sum \partial_p(e) \equiv \sum_{(b_i, e_i) \in \partial_p(e)} b_i p e_i$.

Example 11. *The states of the equation automaton in Example 1 satisfy the following system of equations:*

$$\begin{aligned} e_0 &= t_1 p e_1 \\ e_1 &= 1 p e_2 + t_3 q e_1 + 1 \\ e_2 &= t_2 p e_2 + t_2 t_3 q e_1 + 1 q e_2 + t_2 \end{aligned}$$

For instance, note that $\partial_p(e_2) = \{(t_2, e_2)\}$, $\partial_q(e_2) = \{(t_2 t_3, e_1), (1, e_2)\}$, $\mathsf{out}(e_2) = t_2$, *and* $e_2 = \sum \partial_p(e_2) + \sum \partial_q(e_2) + \mathsf{out}(e_2)$.

Given $e \in \mathsf{Exp}$, we define the *partial derivative* automaton $\mathcal{A}_{\mathsf{pd}}(e) = \langle \{e\} \cup \pi(e), e, \delta_{\mathsf{pd}}, \mathsf{out}(e) \rangle$ where $\delta = \{ \ (e_1, (b, p), e_2) \mid p \in \mathsf{P}, \ (b, e_2) \in \partial_p(e_1) \ \}$.

Lemma 12. *For $e \in \mathsf{Exp}$, $\mathcal{A}_{\mathsf{pd}}(e)$ and $\mathcal{A}_{\mathsf{eq}}(e)$ are identical.*

Proof. The set $\pi(e) = \{e_1, \ldots, e_n\}$ as defined for the equation automata $\mathcal{A}_{\mathsf{eq}}(e)$ [6] satisfies equation (1) and has the following inductive definition: $\pi(p) = \{1\}$, $\pi(b) = \emptyset$, $\pi(e + f) = \pi(e) \cup \pi(f)$, $\pi(e \cdot f) = \pi(e)f \cup \pi(f)$, and $\pi(e^\star) = \pi(e)e^\star$. We have to show that for all $i = 0, \ldots, n$ and $p_r \in \mathsf{P}$, $\sum \partial_{p_r}(e_i) \equiv \sum_{j=1}^{n} b_{ijr} p_r e_j$. These can be proved by induction on the structure of e. □

Proposition 13. $\mathsf{GS}(\mathcal{A}_{\mathsf{pd}}(e)) = \mathsf{GS}(e)$.

Now, it is easy to see that we can define a procedure, HKK, that directly tests the equivalence of any given two KAT expressions. For that, it is enough to modify HKN by taking $(\{e_1\}, \{e_2\})$ as the initial pair and, for $X \subseteq \mathsf{Exp}$, $\tilde{o}_{\mathsf{det}}(X) = \mathsf{out}(X)$, $\Gamma(X, p) = \partial_p(X)$ and $\Delta(X, p) = \Delta_p(X)$.

Algorithm 3: HKK algorithm for KAT expressions.

```
1  def HKK(e1, e2):
2    MAKE({e1}); MAKE({e2})
3    H = ∅
4    UNION({e1},{e2},{e1}); PUSH(H,({e1},{e2}))
5    while (X,Y) = POP(H):
6      if out(X) ≠ out(Y): return False
7      for p ∈ P:
8        B1 = ∂p(X)
9        B2 = ∂p(Y)
10       if ⋃_{(b1,-)∈B1} V_{b1} = ⋃_{(b2,-)∈B2} V_{b2}:
11           for X' ⊆ Δp(X):
12             for Y' ⊆ Δp(Y):
13               if V_{X,p,X'} ∩ V_{Y,p,Y'} ≠ ∅:
14                 X' = FIND(X')
15                 Y' = FIND(Y')
16                 if X' ≠ Y':
17                   UNION(X',Y',Y')
18                   PUSH(H,(X',Y'))
19       else: return False
20   return True
```

7 Conclusions

We considered an automata model for KAT expressions where each transition is labeled by a program symbol and, instead of an atom, a boolean expression. Each transition can, thus, be seen as labeled, in a compact way, by a set of atoms, the ones that satisfy the appropriate boolean expression. Recently, symbolic finite automata (SFA) where transitions are labeled with sets of alphabetic symbols were introduced in order to deal with large alphabets [19]. Although the extension of classical finite automata algorithms to SFAs bears similarities with the ones here presented, SFAs are interpreted over sets of finite words and not over sets of guarded strings. Experiments with the algorithms presented in this paper must be carried out in order to validate their practical applicability and also to suggest goals for a theoretical study of their average-case complexity.

References

[1] Almeida, M., Moreira, N., Reis, R.: Antimirov and Mosses's rewrite system revisited. International Journal of Foundations of Computer Science 20(04), 669 – 684 (2009)

[2] Almeida, M., Moreira, N., Reis, R.: Testing regular languages equivalence. Journal of Automata, Languages and Combinatorics 15(1/2), 7–25 (2010)

[3] Almeida, R., Broda, S., Moreira, N.: Deciding KAT and Hoare logic with derivatives. In: Faella, M., Murano, A. (eds.) Proc. 3rd GANDALF. EPTCS, vol. 96, pp. 127–140 (2012)

[4] Armstrong, A., Struth, G., Weber, T.: Program analysis and verification based on Kleene algebra in Isabelle/HOL. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds.) 4th Inter. Conference ITP 2013, Rennes, France. Proceedings. LNCS, vol. 7998, pp. 197–212. Springer (2013)

[5] Bonchi, F., Pous, D.: Checking NFA equivalence with bisimulations up to congruence. In: Giacobazzi, R., Cousot, R. (eds.) The 40th Annual ACM SIGPLAN-SIGACT Symposium POPL '13. pp. 457–468. ACM (2013)

[6] Broda, S., Machiavelo, A., Moreira, N., Reis, R.: On the average size of Glushkov and equation automata for KAT expressions. In: 19th Inter. Symposium on Fundamentals of Computation Theory. pp. 72–83. No. 8070 in LNCS, Springer (2013)

[7] Champarnaud, J.M., Ziadi, D.: From Mirkin's prebases to Antimirov's word partial derivatives. Fundam. Inform. 45(3), 195–205 (2001)

[8] Cohen, E., Kozen, D., Smith, F.: The complexity of Kleene algebra with tests. Tech. Rep. TR96-1598, Computer Science Department, Cornell University (07 1996)

[9] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. The MIT Press, second edn. (2003)

[10] 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)

[11] Kozen, D.: Kleene algebra with tests. Trans. on Prog. Lang. and Systems 19(3), 427–443 (05 1997)

[12] Kozen, D.: Automata on guarded strings and applications. Matématica Contemporânea 24, 117–139 (2003)

[13] Kozen, D.: On the coalgebraic theory of Kleene algebra with tests. Tech. Rep. `http://hdl.handle.net/1813/10173`, Cornell University (05 2008)

[14] Kozen, D., Smith, F.: Kleene algebra with tests: Completeness and decidability. In: van Dalen, D., Bezem, M. (eds.) Proc. 10th CSL. LNCS, vol. 1258, pp. 244–259. Springer (1996)

[15] Mirkin, B.G.: An algorithm for constructing a base in a language of regular expressions. Engineering Cybernetics 5, 51—57 (1966)

[16] Pereira, D.: Towards Certified Program Logics for the Verification of Imperative Programs. Ph.D. thesis, University of Porto (2013), `http://www.liacc.up.pt/~kat/pdcoq`

[17] Pous, D.: Kleene algebra with tests and Coq tools for While programs. CoRR abs/1302.1737 (2013)

[18] Silva, A.: Position automata for Kleene algebra with tests. Sci. Ann. Comp. Sci. 22(2), 367–394 (2012)

[19] Veanes, M., de Halleux, P., Tillmann, N.: Rex: Symbolic regular expression explorer. In: 3rd Inter. Conference on Software Testing, Verification and Validation, ICST 2010, Paris, France, April 7-9. pp. 498–507. IEEE Computer Society (2010)