Scientific Annals of Computer Science vol. 22 (2), 2012, pp. 367–394
doi: 10.7561/SACS.2012.2.367

Position Automata for Kleene Algebra with Tests

Alexandra SILVA[1]

Abstract

Kleene algebra with tests (KAT) is an equational system that combines Kleene and Boolean algebras. One can model basic programming constructs and assertions in KAT, which allows for its application in compiler optimization, program transformation and dataflow analysis. To provide semantics for KAT expressions, Kozen first introduced *automata on guarded strings*, showing that the regular sets of guarded strings plays the same role in KAT as regular languages play in Kleene algebra. Recently, Kozen described an elegant algorithm, based on "derivatives", to construct a deterministic automaton that accepts the guarded strings denoted by a KAT expression. This algorithm generalizes Brzozowski's algorithm for regular expressions and inherits its inefficiency arising from the explicit computation of derivatives.

In the context of classical regular expressions, many efficient algorithms for compiling expressions to automata have been proposed. One of those algorithms was devised by Berry and Sethi in the 80's (we shall refer to it as Berry-Sethi construction/algorithm, but in the literature it is also referred to as position or Glushkov automata algorithm).

In this paper, we show how the Berry-Sethi algorithm can be used to compile a KAT expression to an automaton on guarded strings. Moreover, we propose a new automata model for KAT expressions and adapt the construction of Berry and Sethi to this new model.

Keywords: position automata, Kleene algebra with tests, coalgebra

[1]Radboud University Nijmegen, The Netherlands. Also affiliated to Centrum Wiskunde & Informatica, Amsterdam, The Netherlands and HASLab / INESC TEC, Universidade do Minho, Portugal (this paper has been written while the author was at the Computer Science Department, Cornell University). Email: alexandra@cs.ru.nl

1 Introduction

Efficient algorithms for compiling regular expressions to deterministic and non-deterministic automata became crucial when regular expressions started to be widely used for pattern matching. One of the most efficient algorithms to translate regular expressions into automata was devised by Berry and Sethi [4].

Kleene algebra with tests (KAT) is an equational system which extends Kleene algebra, the algebra of regular expressions, with a Boolean algebra of tests. KAT expressions are simply regular expressions where the alphabet is two sorted: it contains *actions* and *tests*. The latter must satisfy the axioms of Boolean algebra. This seemingly simple extension allows for a powerful language where modeling and verification of basic programming constructs, such as while loops, conditional tests, Hoare triples, and goto statements, is possible. KAT has successfully been applied in low-level verification tasks involving program transformation, compiler optimization and dataflow analysis [15, 1, 13].

The purpose of this paper is twofold. On the one hand, we observe that the Berry-Sethi algorithm can be directly applied to compile a KAT expression into an equivalent automaton on guarded strings, the original semantic model for KAT proposed by Kozen in [12]. On the other hand, we propose a new automaton model to provide semantics for KAT and adapt the Berry-Sethi algorithm to the new model.

The paper is organized as follows. In Section 2, we start with recalling the main concepts we need from the coalgebraic theory of (non-)deterministic automata and regular expressions. In Section 3, we recall the Berry-Sethi algorithm, from classical regular expressions to non-deterministic automata. In Section 4, we introduce the basics of Kleene algebra with tests (KAT), an extension of the algebra of regular expressions with Boolean tests, and automata on guarded strings (both the deterministic and non-deterministic versions). Section 5 contains the main results of the present paper: we show that the Berry-Sethi construction can be applied directly to KAT yielding a non-deterministic automaton on guarded strings. Moreover, we introduce a new automaton model for KAT which can be regarded as a compromise between the non-deterministic and deterministic versions of Kozen's automata, and adapt the Berry-Sethi construction to compile a KAT expression into this new automata model. The main advantage of the new model is that it will have a smaller number of states than the corresponding automaton on guarded strings. This is important in certain areas of application of KAT, such as verification or model checking of properties.

2 Deterministic Automata and Regular Expressions, Coalgebraically

In this section, we introduce basic notions and notation on deterministic automata and regular expressions. Our presentation is based on the coalgebraic view on automata [18, 19, 20].

2.1 Coalgebra

We will use three notions from coalgebra which we introduce upfront.

Let **Set** be the category of sets and functions. An $\mathcal{F}$-*coalgebra* is a pair $(S, f\colon S \to \mathcal{F}(S))$, where S is a set of states and $\mathcal{F}\colon \mathbf{Set} \to \mathbf{Set}$ is a functor. The functor $\mathcal{F}$, together with the function f, determines the *transition structure* (or dynamics) of the $\mathcal{F}$-coalgebra [19].

An $\mathcal{F}$-*homomorphism* $h\colon (S,f) \to (T,g)$, from an $\mathcal{F}$-coalgebra (S,f) to an $\mathcal{F}$-coalgebra (T,g), is a function $h\colon S \to T$ preserving the transition structure, *i.e.*, such that the following diagram commutes:

$$\begin{tikzcd} S \arrow[r, "h"] \arrow[d, "f"'] & T \arrow[d, "g"] \\ \mathcal{F}(S) \arrow[r, "\mathcal{F}(h)"'] & \mathcal{F}(T) \end{tikzcd} \qquad g \circ h = \mathcal{F}(h) \circ f$$

An $\mathcal{F}$-coalgebra (Ω, ω) is said to be *final* if for any $\mathcal{F}$-coalgebra (S,f) there exists a unique $\mathcal{F}$-homomorphism $\mathbf{beh}_S\colon (S,f) \to (\Omega,\omega)$:

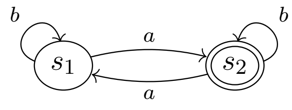
$$\begin{tikzcd} S \arrow[r, dashed, "\mathbf{beh}_S"] \arrow[d, "f"'] & \Omega \arrow[d, "\omega"] \\ \mathcal{F}(S) \arrow[r, dashed, "\mathcal{F}(\mathbf{beh}_S)"'] & \mathcal{F}(\Omega) \end{tikzcd} \qquad \omega \circ \mathbf{beh}_S = \mathcal{F}(\mathbf{beh}_S) \circ f$$

The notion of finality will play a key role later in providing semantics to automata and regular expressions. The functor(s) we consider in the rest of the paper are part of a class for which final coalgebras exist.

2.2 (Non-)Deterministic Automata, Coalgebraically

Let A be a set of input letters (or symbols). A deterministic automaton with inputs in A is a pair $(S, \langle o_S, t_S \rangle)$ consisting of a set of states S and a pair of functions $\langle o_S, t_S \rangle$, where $o\colon S \to 2$ is the *output* function, which determines if a state s is final ($o_S(s) = 1$) or not ($o_S(s) = 0$), and $t\colon S \to S^A$ is the *transition* function[2], which, given an input letter a determines the next state. We will frequently write s_a to denote $t_S(s)(a)$ and refer to s_a as the derivative of s for input a. Moreover, when depicting deterministic automata we will draw a single circle around non-final states and a double circle around final ones.

We illustrate the notation we will use in the representation of deterministic automata in the following example.



$$\begin{array}{lcllcl} o_S(s_1) & = & 0 & o_S(s_2) & = & 1 \\ (s_1)_a & = & s_2 & (s_1)_b & = & s_1 \\ (s_2)_a & = & s_1 & (s_2)_b & = & s_2 \end{array}$$

Deterministic automata are coalgebras for the functor $\mathcal{D}(X) = 2 \times X^A$. The classical notion of automata homomorphism will instantiate precisely to the definition of coalgebra homomorphism for the functor $\mathcal{D}$: given two deterministic automata $(S, \langle o_S, t_s \rangle)$ and $(T, \langle o_T, t_T \rangle)$, a function $h\colon S \to T$ is a homomorphism if it preserves outputs and input derivatives, that is $o_T(h(s)) = o_S(s)$ and $h(s)_a = t_T(h(s))(a) = h^A(t_S(s))(a) = h(s_a)$, for all $a \in A$. These equations correspond to the commutativity of the following diagram.

$$\begin{tikzcd} S \arrow[r, "h"] \arrow[d, "\langle o_S, t_S \rangle"'] & T \arrow[d, "\langle o_T, t_T \rangle"] \\ 2 \times S^A \arrow[r, "id \times h^A"'] & 2 \times T^A \end{tikzcd}$$

The input derivative s_a of a state s for input $a \in A$ can be extended to the word derivative s_w of a state s for input $w \in A^*$ by defining, by induction on the length of w, $s_\epsilon = s$ and $s_{aw'} = (s_a)_{w'}$, where ϵ denotes the empty word and aw' the word obtained by prefixing w' with the letter a. This enables an

[2]Here, and in the sequel, we represent the transition function of an automaton curried. Alternatively, and equivalently, the function could be represented as $t\colon S \times A \to S$ or $t\colon S \to (A \to S)$. The curried representation is more amenable for the coalgebraic treatment of automata.

easy definition of the semantics of a state s of a deterministic automaton - the language $L(s) \in 2^{A^*}$ of a state s is given by the following characteristic function:

$$L(s)(w) = o_S(s_w) \tag{1}$$

The language recognized by s contains all words w for which $L(s)(w) = 1$. For instance, the language recognized by state s_1 of the automaton above is the set of all words with an odd number of a's. It is easy to check that, for example, $L(s_1)(bab) = o_S((s_1)_{bab}) = o_S(s_2) = 1$ and $L(s_1)(aab) = o_S((s_1)_{aab}) = o_S(s_1) = 0$.

Given two deterministic automata $(S, \langle o_S, t_S \rangle)$ and $(T, \langle o_T, t_T \rangle)$ a relation $R \subseteq S \times T$ is a bisimulation if $\langle s, t \rangle \in R$ implies

$$o_S(s) = o_T(t) \quad \text{and} \quad \langle s_a, t_a \rangle \in R \text{ for all } a \in A$$

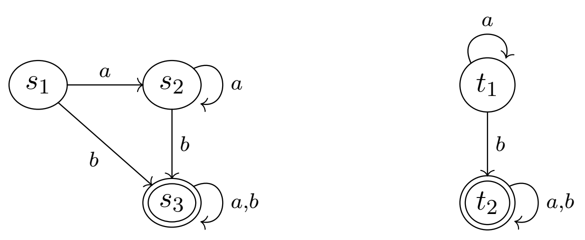
We will write $s \sim t$ whenever there exists a bisimulation R containing $\langle s, t \rangle$. This concrete definition of bisimulation can be recovered as a special case of the general definition of bisimulation for $\mathcal{F}$-coalgebras [19] by instantiating the functor $\mathcal{F}$ to $\mathcal{D}(X) = 2 \times X^A$. The following theorem guarantees that the definition above is a valid proof principle for language equivalence of deterministic automata. We omit the proof here, for details see [18].

Theorem 1 (Coinduction) *Given two deterministic automata $(S, \langle o_S, t_S \rangle)$ and $(T, \langle o_T, t_T \rangle)$, $s \in S$ and $t \in T$:*

$$s \sim t \Rightarrow L(s) = L(t)$$

To determine whether two states s and t of two deterministic automata $(S, \langle o_S, t_S \rangle)$ and $(T, \langle o_T, t_T \rangle)$ (over the same alphabet) recognize the same language we can now use coinduction: it is enough to construct a bisimulation containing $\langle s, t \rangle$.

Example 1 *Let $(S, \langle o_S, t_S \rangle)$ and $(T, \langle o_T, t_T \rangle)$ be the deterministic automata over the alphabet $\{a, b\}$ given by*



The relation $R = \{\langle s_1, t_1\rangle, \langle s_2, t_1\rangle, \langle s_3, t_2\rangle\}$ *is a bisimulation:*

$$\begin{array}{ll} o_S(s_1) = o_S(s_2) = 0 = o_T(t_1) & o_S(s_3) = 1 = o_T(t_2) \\ (s_1)_a = s_2 \; R \; t_1 = (t_1)_a & (s_1)_b = s_3 \; R \; t_2 = (t_1)_b \\ (s_2)_a = s_2 \; R \; t_1 = (t_1)_a & (s_2)_b = s_3 \; R \; t_2 = (t_1)_b \\ (s_3)_a = s_3 \; R \; t_2 = (t_2)_a & (s_3)_b = s_3 \; R \; t_2 = (t_2)_b \end{array}$$

Thus, $L(s_1) = L(t_1) = L(s_2)$ *and* $L(s_3) = L(t_3)$.

The language recognized by a state s is the behavior (or semantics) of s. Thus, the set of languages 2^{A^*} over A can be thought of as the universe of all possible behaviors for deterministic automata. We now turn 2^{A^*} into a deterministic automaton (with inputs in A) and then show that such an automaton has the universal property of being *final*, which will connect the coalgebraic semantics induced by the functor with the classical language semantics we have just presented.

For an input letter $a \in A$, the input derivative K_a of a language $K \in 2^{A^*}$ on input a is defined by $K_a(w) = L(aw)$. The output of K is defined by $K(\epsilon)$. These notions determine a deterministic automaton $(2^{A^*}, \langle o_L, t_L\rangle)$ defined, for $K \in 2^{A^*}$ and $a \in A$, by $o_L(K) = K(\epsilon)$ and $t_L(K)(a) = K_a$.

Theorem 2 *The automaton* $(2^{A^*}, \langle o_L, t_L\rangle)$ *is final. That is, for any deterministic automaton* $(S, \langle o_S, t_S\rangle)$, $L\colon S \to 2^{A^*}$ *is the unique homomorphism which makes the following diagram commute.*

$$\begin{tikzcd} S \arrow[r, dashed, "L"] \arrow[d, "\langle o_S, t_S\rangle"'] & 2^{A^*} \arrow[d, "\langle o_L, t_L\rangle"] \\ 2 \times S^A \arrow[r, dashed, "id \times L^A"'] & 2 \times (2^{A^*})^A \end{tikzcd}$$

Given a state s*, the language* $L(s)$ *is precisely the language recognized by* s *(as defined in equation (1)).*

Proof: We have to prove that the diagram commutes and that L is unique. First the commutativity:

$$\begin{array}{l} o_L(L(s)) = L(s)(\epsilon) = o_S(s) \\ t_L(L(s))(a) = L(s)_a = \lambda w.L(s)(aw) = \lambda w.L(s_a)(w) = L(s_a) = L(t_S(s))(a) \end{array}$$

For the one but last step, note that, by definition of L (equation (1))

$$L(s)(aw) = o_S(s_{aw}) = o_S((s_a)_w) = L(s_a)(w)$$

For the uniqueness, suppose there is another morphism $h\colon S \to 2^{A^*}$ such that, for every $s \in S$ and $a \in A$, $o_L(h(s)) = o_S(s)$ and $h(s)_a = h(s_a)$.

We prove by induction on the length of words $w \in A^*$ that $h(s) = L(s)$.

$$\begin{aligned}
&h(s)(\epsilon) = o_L(h(s)) = o_S(s) = L(s)(\epsilon)\\
&h(s)(aw) = (h(s)_a)(w) = h(s_a)(w) \overset{(IH)}{=} L(s_a)(w) = L(s)(aw)
\end{aligned}$$

□

The semantics induced by the unique map L into the final coalgebra coincides, in the case of deterministic automata, with the bisimulation semantics we defined above, that is $s \sim t \Leftrightarrow L(s) = L(t)$ (the implication in Theorem 1 is actually an equivalence).

Moore automata are a slight variation on deterministic automata. More precisely, the codomain of the output function is changed from 2 to B. Formally, a Moore automaton with inputs in A and outputs in B is a pair $(S, \langle o, t\rangle)$ where S is a set of states, $o\colon S \to B$ is the output function and $t\colon S \to S^A$ is the transition function.

Similarly to the change on the output function, the carrier set of the final coalgebra for Moore automata is B^{A^*} (in contrast to 2^{A^*}), the coalgebra structure $\langle o_M, t_M\rangle\colon B^{A^*} \to B \times (B^{A^*})^A$ is defined similarly

$$o_M(f) = f(\varepsilon) \qquad\qquad t_M(f)(a)(w) = f(aw)$$

and we have the following finality result (proof similar to Theorem 2).

Theorem 3 *The automaton $(B^{A^*}, \langle o_M, t_M\rangle)$ is final. That is, for any Moore automaton $(S, \langle o_S, t_S\rangle)$, there is a unique homomorphism which makes the following diagram commute.*

$$\begin{tikzcd}
S \arrow[r, dashed, "h"] \arrow[d, "\langle o_S, t_S\rangle"'] & B^{A^*} \arrow[d, "\langle o_M, t_M\rangle"] \\
B \times S^A \arrow[r, dashed, "id \times h^A"'] & B \times (B^{A^*})^A
\end{tikzcd}$$

A non-deterministic automaton (NDA) is similar to a deterministic automaton but the transition function gives a set of next-states for each input letter instead of a single state. NDA's often provide more compact representations of regular languages than deterministic automata. For that, they are computationally very interesting and much research has been devoted to constructions compiling a regular expression into an NDA [2, 8, 4, 21, 16, 7] (we will show an example of such construction below). Surprisingly, in what concerns language acceptance NDA's are not more powerful than deterministic automata. A state s of an NDA accepts a word if there is a path starting in s labeled by w which leads to a final state. For every NDA there exists a deterministic automaton with a state equivalent to a given state of the NDA. Such deterministic automaton can be obtained from a given NDA by the so-called subset (or powerset) construction first introduced by Rabin and Scott [17], which we will show below coalgebraically.

Formally, an NDA over the input alphabet A is a pair $(S, \langle o, \delta\rangle)$, where S is a set of states and $\langle o, \delta\rangle \colon S \to 2 \times (\mathcal{P}_\omega(S))^A$ is a function pair with o as before and where δ determines for each input letter a a finite set of possible next states.

As an example of the compactness of NDA's, consider the following regular language (taken from [11]):

$$\{w \in \{a,b\}^* \mid \text{the fifth symbol from the right is } a\}$$

One can intuitively construct an NDA with a state s, having two outgoing a-transitions, which recognizes this language (which could be, for instance, denoted by the regular expression $(a+b)^*a(a+b)(a+b)(a+b)(a+b)$):

$$\begin{tikzcd} s \arrow[loop above, "{a,b}"] \arrow[r, "a"] & 5 \arrow[r, "{a,b}"] & 4 \arrow[r, "{a,b}"] & 3 \arrow[r, "{a,b}"] & 2 \arrow[r, "{a,b}"] & 1 \end{tikzcd}$$

A deterministic automaton recognizing the same language will have at least $2^5 = 32$ states.

In order to formally compute the language recognized by a state x of an NDA $\mathcal{A}$, it is usual to first determinize it, constructing a deterministic automaton $\mathbf{det}(\mathcal{A})$ where the state space is $\mathcal{P}_\omega(S)$, and then compute the language recognized by the state $\{x\}$ of $\mathbf{det}(\mathcal{A})$. Next, we describe in coalgebraic terms how to construct the automaton $\mathbf{det}(\mathcal{A})$ [19].

Given an NDA $\mathcal{A} = (S, \langle o, \delta\rangle)$, we construct $\mathbf{det}(\mathcal{A}) = (\mathcal{P}_\omega(S), \langle \overline{o}, \overline{\delta}\rangle)$, where, for all $Y \in \mathcal{P}_\omega(S)$, $a \in A$, the functions $\overline{o} \colon \mathcal{P}_\omega(S) \to 2$ and $\overline{\delta} \colon \mathcal{P}_\omega(S) \to \mathcal{P}_\omega(S)^A$ are

$$\overline{o}(Y) = \begin{cases} 1 & \text{if } \exists_{y \in Y}\, o(y) = 1 \\ 0 & \text{otherwise} \end{cases} \qquad \overline{\delta}(Y)(a) = \bigcup_{y \in Y} \delta(y)(a).$$

The automaton $\mathbf{det}(\mathcal{A})$ is such that the language $L(\{x\})$ recognized by $\{x\}$ is the same as the one recognized by x in the original NDA $\mathcal{A}$ (more generally, the language recognized by state $X \in \mathcal{P}_\omega(S)$ of $\mathbf{det}(\mathcal{A})$ is the union of the languages recognized by each state $x \in X$ of $\mathcal{A}$).

We summarize the situation above with the following commuting diagram:

$$\begin{tikzcd}
S \arrow[r, "\{\cdot\}"] \arrow[d, "\langle o,\delta\rangle"'] & \mathcal{P}_\omega(S) \arrow[r, dashed, "L"] \arrow[dl, "\langle \overline{o},\overline{\delta}\rangle"] & 2^{A^*} \arrow[d, "\langle o_L,t_L\rangle"] \\
2 \times \mathcal{P}_\omega(S)^A \arrow[rr, dashed] & & 2 \times (2^{A^*})^A
\end{tikzcd}$$

We note that the language semantics of NDA's, presented in the above diagram, can alternatively be achieved by using λ-coinduction [3, 9].

2.3 Regular Expressions

We will now recall the basic definitions and results on regular expressions. The set $\mathcal{R}(A)$ of regular expressions over a finite input alphabet A is given by the following syntax:

$$r, r_1, r_2 ::= \underline{1} \mid \underline{0} \mid a \in A \mid r_1 + r_2 \mid r_1 r_2 \mid r^*$$

The semantics of regular expressions is given in terms of languages[3] and it is defined as a map $L\colon \mathcal{R}(A) \to \mathcal{P}(A^*)$ by induction on the syntax as follows:

$$\begin{array}{lll} L(\underline{1}) = \{\epsilon\} & L(\underline{0}) = \emptyset & L(a) = \{a\} \\ L(r_1 + r_2) = L(r_1) \cup L(r_2) & L(r_1 r_2) = L(r_1) \cdot L(r_2) & L(r^*) = L(r)^* \end{array} \tag{2}$$

where, given languages l, l_1 and l_2, $l_1 \cdot l_2 = \{w_1 w_2 \mid w_1 \in l_1 \text{ and } w_2 \in l_2\}$; $l^* = \bigcup_{n\in\mathbb{N}} l^n$, and, for $n \in \mathbb{N}$, l^n is inductively defined by $l^0 = \{\epsilon\}$ and $l^{n+1} = l \cdot l^n$.

Here, we have intentionally reused $L(r)$ to represent the language denoted by a regular expression $r \in \mathcal{R}(A)$ (recall that we had used $L(s)$ to represent the language recognized by a state s of a deterministic automaton). This is because we know from Kleene's theorem that finite deterministic automata recognize precisely the languages denoted by regular expressions.

We now equip the set $\mathcal{R}(A)$ with a deterministic automaton structure. This definition was first proposed by Brzozowski in his paper *Derivatives of*

[3]Here, we represent languages as subsets of A^*, rather than functions 2^{A^*}. Although we prefer the latter view on languages, the traditional semantics of regular expressions was presented as sets of words and we recall it here unchanged. We will only use the set interpretation on languages when referring to the classical semantics of regular expressions.

regular expressions [5] and, for that reason, it is occasionally referred to as Brzozowski derivatives. We define the output $o_{\mathcal{R}}(r)$ of a regular expression r by

$$\begin{array}{rclcrcl} o_{\mathcal{R}}(\underline{0}) & = & 0 & \qquad & o_{\mathcal{R}}(r_1 + r_2) & = & o_{\mathcal{R}}(r_1) \vee o_{\mathcal{R}}(r_2) \\ o_{\mathcal{R}}(\underline{1}) & = & 1 & & o_{\mathcal{R}}(r_1 r_2) & = & o_{\mathcal{R}}(r_1) \wedge o_{\mathcal{R}}(r_2) \\ o_{\mathcal{R}}(a) & = & 0 & & o_{\mathcal{R}}(r^*) & = & 1 \end{array}$$

and the input derivative $t_{\mathcal{R}}(r)(a) = r_a$ by

$$\begin{array}{rclcrcl} (\underline{0})_a & = & \underline{0} & \qquad & (r_1 + r_2)_a & = & (r_1)_a + (r_2)_a \\ (\underline{1})_a & = & \underline{0} & & (r_1 r_2)_a & = & \begin{cases} (r_1)_a r_2 & \text{if } o_{\mathcal{R}}(r_1) = 0 \\ (r_1)_a r_2 + (r_2)_a & \text{otherwise} \end{cases} \\ (a)_{a'} & = & \begin{cases} \underline{1} & \text{if } a = a' \\ \underline{0} & \text{if } a \neq a' \end{cases} & & (r^*)_a & = & r_a r^* \end{array}$$

In the definition of $o_{\mathcal{R}}$ we use the fact that $2 = \{0, 1\}$ can be given a lattice structure $(\{0, 1\}, \vee, \wedge, 0, 1)$ (0 is neutral with respect to $\vee$ and 1 with respect to $\wedge$).

Intuitively, for a regular expression r, $o_{\mathcal{R}}(r) = 1$ if the language denoted by r contains the empty word ϵ and $o_{\mathcal{R}}(r) = 0$ otherwise. The regular expression r_a denotes the language containing all words w such that aw is in the language denoted by r.

Similarly to what happened in deterministic automata, the input derivative r_a of a regular expression r for input a can be extended to the word derivative r_w of r for input $w \in A^*$ by defining $r_\epsilon = r$ and $r_{aw} = (r_a)_w$.

We have now defined a deterministic automaton $(\mathcal{R}(A), \langle o_{\mathcal{R}}, t_{\mathcal{R}} \rangle)$ and thus, by Theorem 2, we have a unique map L which makes the following diagram commute.

$$\begin{tikzcd} \mathcal{R}(A) \arrow[r, dashed, "L"] \arrow[d, "\langle o_{\mathcal{R}}, t_{\mathcal{R}} \rangle"'] & 2^{A^*} \arrow[d, "\langle o_L, t_L \rangle"] \\ 2 \times (\mathcal{R}(A))^A \arrow[r, dashed, "id \times L^A"'] & 2 \times (2^{A^*})^A \end{tikzcd} \qquad (3)$$

We now prove that, for any $r \in \mathcal{R}(A)$, the semantics defined inductively in (2) is the same as the one given by the unique map into the final coalgebra $L \colon \mathcal{R}(A) \to 2^{A^*}$.

Theorem 4 *For all $r \in \mathcal{R}(A)$ and $w \in A^*$,*

$$w \in L(r) \Leftrightarrow o_{\mathcal{R}}(r_w) = 1$$

where $L(r)$ is the inductively defined semantics from equation (2).

Proof: By induction on the structure of r.

$$\begin{aligned}
&L(\underline{0}) = \emptyset \text{ and } o_{\mathcal{R}}((\underline{\emptyset})_w) = 1\\
&L(\underline{1}) = \{\epsilon\} \text{ and } o_{\mathcal{R}}((\underline{1})_w) = 1 \Leftrightarrow w = \epsilon\\
&L(a) = \{a\} \text{ and } o_{\mathcal{R}}((a)_w) = 1 \Leftrightarrow w = a\\
&w \in L(r_1 + r_2) \Leftrightarrow w \in L(r_1) \text{ or } w \in L(r_2)\\
&\qquad\overset{(IH)}{\Leftrightarrow} o_{\mathcal{R}}((r_1)_w) = 1 \text{ or } o_{\mathcal{R}}((r_2)_w) = 1 \Leftrightarrow o_{\mathcal{R}}((r_1 + r_2)_w) = 1\\
&w \in L(r_1 r_2) \Leftrightarrow w = w_1 w_2, \text{ with } w_1 \in L(r_1) \text{ and } w_2 \in L(r_2)\\
&\qquad\overset{(IH)}{\Leftrightarrow} o_{\mathcal{R}}((r_1)_{w_1}) = 1 \text{ and } o_{\mathcal{R}}((r_2)_{w_2}) = 1 \Leftrightarrow o_{\mathcal{R}}((r_1 r_2)_w) = 1\\
&w \in L(r^*) \Leftrightarrow w \in L(r)^n, \text{ for some } n \in \mathbb{N} \Leftrightarrow w = w_1 \ldots w_n \text{ with } w_i \in L(r)\\
&\qquad\overset{(IH)}{\Leftrightarrow} w = w_1 \ldots w_n \text{ with } o_{\mathcal{R}}(r_{w_i}) = 1 \Leftrightarrow o_{\mathcal{R}}((r^*)_w) = 1
\end{aligned}$$

□

We have now proved that the classical semantics of both deterministic automata and regular expressions coincides with the coalgebraic semantics. In the sequel, we will say that a regular expression r and a state s of a deterministic automaton are equivalent if $L(s) = L(r)$.

3 From Regular Expressions to Non-Deterministic Automata: the Berry-Sethi Construction

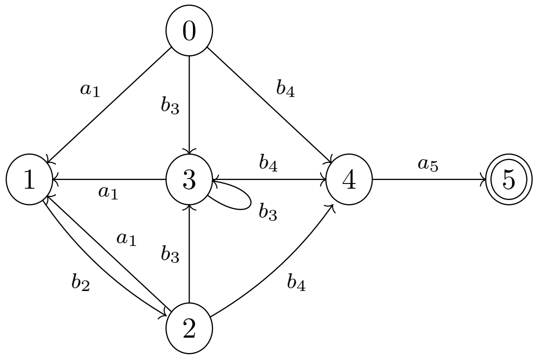
There are several algorithms to construct a non-deterministic automaton from a regular expression. We will show here the one presented in [4] by Berry and Sethi. We shall generalize this algorithm in the next section in order to deal with the expressions of Kleene algebra with tests. The basic idea behind the algorithm is that of marking: all input letters in a regular expression are marked (with subscripts) in order to make them distinct. As an example, a marked version of $(ab+b)^*ba$ is $(a_1b_2+b_3)^*b_4a_5$, where a_1 and a_5 are considered different letters. The choice we made for the subscripts are the positions of the letters in the expression. For that reason the Berry-Sethi construction is often referred to as position automaton.

We will explain the algorithm with an example (taken from [4]) and then state the results that justify its correctness.

Example 2 *Let* $r = (ab + b)^*ba$ *and let* $\overline{r} = (a_1b_2 + b_3)^*b_4a_5$ *be its marked version. We define* $c_i = (\overline{r})_w$*, for* w *a prefix of length* i *of* $a_1b_2b_3b_4a_5$*, and call it the continuation* i *of* $\overline{r}$*. We then construct an automaton from* $\overline{r}$ *in the following way:*

1. *The automaton will have a state* $i \in \{1, 2, 3, 4, 5\}$ *for each distinct symbol in* $\overline{r}$ *plus an extra state* 0 *that will be language equivalent*[4] *to* $\overline{r}$*.*
2. *A state* i *has a transition to state* j*, labeled by* a_j*, if* $(c_i)_{a_j} = c_j$*. A state* i *is final if* $o_{\mathcal{R}}(c_i) = 1$*.*

The automaton resulting from $\overline{r} = (a_1b_2 + b_3)^*b_4a_5$ *is the following*

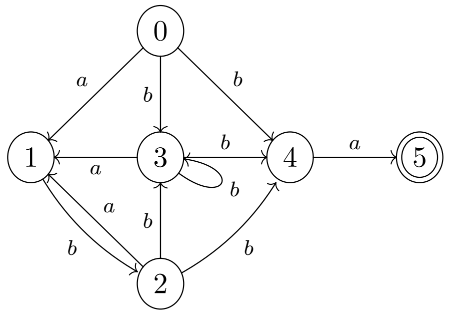


$$\begin{aligned}
c_0 &= (\overline{r})_\epsilon = (a_1b_2 + b_3)^*b_4a_5 & c_1 &= (\overline{r})_{a_1} = b_2(a_1b_2 + b_3)^*b_4a_5 \\
c_2 &= (\overline{r})_{a_1b_2} = (c_1)_{b_2} = (a_1b_2 + b_3)^*b_4a_5 & c_4 &= (\overline{r})_{a_1b_2b_3b_4} = (c_3)_{b_4} = a_5 \\
c_3 &= (\overline{r})_{a_1b_2b_3} = (c_2)_{b_3} = (a_1b_2 + b_3)^*b_4a_5 & c_5 &= (\overline{r})_{a_1b_2b_3b_4a_5} = (c_4)_{a_5} = \underline{1}
\end{aligned}$$

Note that to compute the transition structure we had to compute all input derivatives for each c_i*. This can be overcome by using some of the properties of derivatives of expressions with distinct symbols (more below). Now, note that by deleting all the marks in the labels of the automaton above the state* 0 *of the resulting NDA accepts precisely the language denoted by* $(ab + b)^*ba$

[4]Here, by language equivalent we mean that the state 0 recognizes the same language that the expression denotes. More precisely, $L(\{0\}) = L(\bar{r})$.

(all words that finish with ba and all other occurrences of a are followed by one or more b's).



(4)

The algorithm above works as expected due to the properties of derivatives of expressions with distinct letters. We summarize the crucial properties for the correctness of the algorithm.

Theorem 5 ([4, Proposition 3.2 and Theorem 3.4]) *Let $\overline{r}$ be the regular expression obtained from r by marking all symbols to make them distinct. Then, the following holds:*

1. *If $\mathcal{A}'$ is an automaton with a state s such that $L(s) = L(\overline{r})$, then the state s of the automaton $\mathcal{A}$, obtained from $\mathcal{A}'$ by unmarking all the labels, is such that $L(s) = L(r)$.*

2. *Given any symbol a and word w, the derivative $(\overline{r})_{aw}$ is either $\underline{0}$ or unique modulo associativity, commutativity and idempotency.*

Starting from a regular expression $r \in \mathcal{R}(A)$, we can then obtain a nondeterministic automaton by first marking the symbols, then applying the algorithm above and finally unmarking the labels. If wanted, a deterministic automaton can then be obtained via the subset construction (the complexity of this construction for position automata was studied in [6]).

In [4], the authors presented also a more efficient way of computing the position automaton, based on the fact that each continuation is uniquely determined by an input symbol. We briefly recall it here, since this is precisely the version we will later generalize for `KAT` expressions. Let $pos(r)$ denote the positions (distinct symbols) in the regular expression r. For any regular expression r and $i \in pos(r)$ we define:

$$\begin{array}{lcl} first(r) & = & \{i \mid p_i w \in L(\overline{r}), w \in A^*, p_i \in A\} \\ follow(r,i) & = & \{j \mid u p_i p_j v \in L(\overline{r}), u.v \in A^*, p_i, p_j \in A\} \\ last(r) & = & \{i \mid w p_i \in L(\overline{r}), w \in A^*, p_i \in A\} \end{array}$$

The set *first*(r) contains all indexes i of the letters p_i that can appear in the beginning of a word in the language $L(\bar{r})$. For instance, in the expression $\bar{r} = (a_1b_2 + b_3)^*b_4a_5$ above *first*$(r) = \{1, 3, 4\}$. Dually, *last*(r) contains all indexes of letters that can appear at the end of a word in $L(\bar{r})$. In the example, *last*$(r) = \{5\}$. The set *follow*(r, i) has all the indexes of letters that can follow letter p_i in a word in the language $L(\bar{r})$. For instance, in our example, *follow*$(r, 2) = \{1, 3, 4\}$.

These sets can be computed efficiently from the expression: we recall [4, Proposition 4.3].

Proposition 1 ([4, Proposition 4.3]) *Let r be a regular expression with distinct symbols. $\mathbf{F}$, defined by the rules below, is such that $\mathbf{F}(r, \{!\})$ yields a set of pairs of the form $\langle a_i, follow(r!, i)\rangle$, where ! is a symbol distinct from all symbols in r. The rules are:*

$$\begin{array}{lcl}
\mathbf{F}(r_1 + r_2, S) & = & \mathbf{F}(r_1, S) \cup \mathbf{F}(r_2, S) \\
\mathbf{F}(r_1 r_2, S) & = & \mathbf{F}(r_1, \mathit{first}(r_2) \cup o_{\mathcal{R}}(r_2).S) \cup \mathbf{F}(r_2, S) \\
\mathbf{F}(r_1^*, S) & = & \mathbf{F}(r_1, \mathit{first}(r_1) \cup S) \\
\mathbf{F}(a, S) & = & \{\langle a, S\rangle\} \\
\mathbf{F}(\underline{1}, S) & = & \mathbf{F}(\underline{0}, S) = \emptyset
\end{array}$$

Here, for a set S, $\underline{1}.S = S$ and $\underline{0}.S = \emptyset$. Note that in $\mathbf{F}$ also the set last(r) is computed: $i \in last(r) \Leftrightarrow ! \in follow(r!, i)$.

The position automaton corresponding to a given regular expression $r \in \mathcal{R}(A)$ is then given by

$$\mathcal{A}_{pos}(r) = (\{0\} \cup pos(\overline{r}), \langle o, \delta\rangle)$$

where $\overline{r}$ is the marked version of r and o and δ are defined as follows:

$$o(0) = o_{\mathcal{R}}(r) \qquad\qquad o(i) = \begin{cases} 1 & \text{if } i \in last(\overline{r}) \\ 0 & \text{otherwise} \end{cases}$$

$$\delta(0)(a) = \{j \mid j \in \mathit{first}(\overline{r}), unmark(a_j) = a\}$$

$$\delta(i)(a) = \{j \mid j \in follow(\overline{r}, i), unmark(a_j) = a\} \qquad i \neq 0$$

We show an example of the algorithm above. We consider again $r = (ab+b)^*ba$ and its marked version $\overline{r} = (a_1b_2+b_3)^*b_4a_5$.

$$\begin{array}{lcll} \mathit{first}(\overline{r}) & = & \{1,3,4\} & \mathit{first}(a_1b_2+b_3) = \{1,3\} \\ \mathit{first}(a_1b_2) & = & \{1\} & \mathit{first}(b_4a_5) = \{4\} \end{array}$$

$$\begin{array}{ll} & \mathbf{F}(\overline{r},\{!\}) \\ = & \mathbf{F}((a_1b_2+b_3)^*,\{4\}) \cup \mathbf{F}(b_4a_5,\{!\}) \\ = & \mathbf{F}(a_1b_2+b_3,\{1,3,4\}) \cup \mathbf{F}(b_4,\{5\}) \cup \mathbf{F}(a_5,\{!\}) \\ = & \mathbf{F}(a_1b_2,\{1,3,4\}) \cup \mathbf{F}(b_3,\{1,3,4\}) \cup \{\langle b_4,\{5\}\rangle, \langle a_5,\{!\}\rangle\} \\ = & \mathbf{F}(a_1,\{2\}) \cup \mathbf{F}(b_2,\{1,3,4\}) \cup \{\langle b_3,\{1,3,4\}\rangle, \langle b_4,\{5\}\rangle, \langle a_5,\{!\}\rangle\} \\ = & \{\langle a_1,\{2\}\rangle, \langle b_2,\{1,3,4\}\rangle, \langle b_3,\{1,3,4\}\rangle, \langle b_4,\{5\}\rangle, \langle a_5,\{!\}\rangle\} \end{array}$$

The position automaton $\mathcal{A}_{pos}(r)$ constructed is the same as the one presented above in (4).

It should be remarked that the construction of the position automaton from a regular expression does not always extend to additional operators, such as intersection or complement. This is a disadvantage when compared, for instance, to the algorithm based on Brzozowski derivatives.

4 Automata on Guarded Strings and `KAT` Expressions

Kleene algebra with tests (`KAT`) is an equational system that combines Kleene and Boolean algebra. One can model basic programming constructs and assertions in `KAT`, which allows for its application in compiler optimization, program transformation or dataflow analysis [15, 1, 13]. In this section, we will recall the basic definitions of `KAT` and we will show how to generalize the Berry-Sethi construction (Section 3) in order to (efficiently) obtain an automaton from a `KAT` expression.

Definition 1 (Kleene algebra with tests) *A Kleene algebra with tests is a two-sorted structure* $(\Sigma, B, +, \cdot, (-)^*, ^-, 0, 1)$ *where*

- $(\Sigma, +, \cdot, (-)^*, 0, 1)$ *is a Kleene algebra,*
- $(B, +, \cdot, ^-, 0, 1)$ *is a Boolean algebra, and*
- $(B, +, \cdot, ^-, 0, 1)$ *is a subalgebra of* $(\Sigma, +, ., (-)^*, 0, 1)$.

The operator $^-$ *denotes negation.*

Given a set $\mathtt{P}$ of (primitive) action symbols and a set $\mathtt{B}$ of (primitive) test symbols, we can define the free Kleene algebra with tests on generators $\mathtt{P} \cup \mathtt{B}$ as follows. Syntactically, the set *BExp* of Boolean tests is given by:

$$\mathit{BExp} \ni b ::= \mathtt{b} \in \mathtt{B} \mid b_1 b_2 \mid b_1 + b_2 \mid \bar{b} \mid \mathtt{0} \mid \mathtt{1}$$

The set of $\mathtt{KAT}$ expressions is given by

$$\mathtt{Exp} \ni e, f ::= \mathtt{p} \in \mathtt{P} \mid b \in \mathit{BExp} \mid ef \mid e + f \mid (e)^*$$

The free Kleene algebra with tests on generators $\mathtt{P} \cup \mathtt{B}$ is obtained by quotienting *BExp* by the axioms of Boolean algebra and $\mathtt{Exp}$ by the axioms of Kleene algebra. We will use below the natural order on expressions: $e \leq f \iff e + f \equiv f$, where $\equiv$ denotes equivalence provable using the axioms of Kleene/Boolean algebra (these include axioms for idempotency, associativity and commutativity of $+$, among others, for more details see, for instance, [12]).

Guarded strings were introduced in [10] as an abstract interpretation for program schemes. They are like ordinary strings over an input alphabet $\mathtt{P}$, but the symbols in $\mathtt{P}$ alternate with the atoms of the free Boolean algebra generated by $\mathtt{B}$. The set $\mathtt{At}$ of atoms is given by $\mathtt{At} = 2^{\mathtt{B}}$. We define the set $\mathtt{GS}$ of guarded strings by

$$\mathtt{GS} = (\mathtt{At} \times \mathtt{P})^* \mathtt{At}$$

Kozen [12] showed that the regular sets of guarded strings play the same role in $\mathtt{KAT}$ as regular languages play in Kleene algebra (both sets are actually the final coalgebra of a given functor). He showed an analogue of Kleene's theorem: *automata on guarded strings*, which are non-deterministic automata over the alphabet $\mathtt{P} \cup \mathtt{B}$, recognize precisely the regular sets of guarded strings.

Definition 2 (Regular sets of guarded strings) *Each $\mathtt{KAT}$ expression e denotes a set $G(e)$ of guarded strings defined inductively on the structure of e as follows:*

$$\begin{array}{lcl} G(\mathtt{p}) & = & \{\alpha \mathtt{p} \beta \mid \alpha, \beta \in \mathtt{At}\} \\ G(b) & = & \{\alpha \mid \alpha \leq b\} \\ G(e+f) & = & G(e) \cup G(f) \\ G(ef) & = & G(e) \diamond G(f) \\ G(e^*) & = & \bigcup_{n \geq 0} G(e)^n \end{array}$$

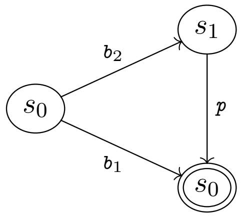
where, given two guarded strings $x = \alpha_0 \mathtt{p}_0 \ldots \mathtt{p}_{n-1}\alpha_n$ and $y = \beta_0 \mathtt{q}_1 \ldots \mathtt{q}_{n-1}\beta_n$, we define the fusion product of x and y by $x \diamond y = \alpha_0 \mathtt{p}_0 \ldots \mathtt{p}_n \alpha_n \mathtt{q}_1 \ldots \mathtt{q}_{n-1}\beta_n$, if $\alpha_n = \beta_0$, otherwise $x \diamond y$ is undefined. Then, given $X, Y \subseteq \mathtt{GS}$, $X \diamond Y$ is the set containing all existing fusion products $x \diamond y$ of $x \in X$ and $y \in Y$ and X^n is defined inductively as $X^0 = X$ and $X^{n+1} = X \diamond X^n$.

A set of guarded strings is regular if it is equal to $G(e)$ for some $\mathtt{KAT}$ expression e. Note that a guarded string is itself a $\mathtt{KAT}$ expression and $G(x) = \{x\}$.

Example 3 *Consider the $\mathtt{KAT}$ expression $e = \mathtt{b}_1 + \mathtt{b}_2\mathtt{p}$ over $\mathtt{B} = \{\mathtt{b}_1, \mathtt{b}_2\}$ and $\mathtt{P} = \{\mathtt{p}\}$. We compute the set $G(e)$:*

$$\begin{aligned} G(e) &= G(\mathtt{b}_1) \cup (G(\mathtt{b}_2) \diamond G(\mathtt{p})) \\ &= \{\alpha \mid \alpha \leq \mathtt{b}_1\} \cup (\{\alpha \mid \alpha \leq \mathtt{b}_2\} \diamond \{\alpha \mathtt{p} \beta \mid \alpha, \beta \in \mathtt{At}\}) \\ &= \{\alpha \mid \alpha \leq \mathtt{b}_1\} \cup \{\alpha \mathtt{p} \beta \mid \alpha \leq \mathtt{b}_2, \beta \in \mathtt{At}\} \end{aligned}$$

We will now show an example of an automaton on guarded strings. As mentioned above such automaton is just a non-deterministic automaton over the alphabet $A = \mathtt{P} \cup \mathtt{B}$, that is $(S, \langle o_S, t_S \rangle)$ with $o \colon S \to 2$ and $t \colon S \to \mathcal{P}_\omega(S)^A$. State s_0 of the following automaton would recognize *(we shall explain the precise meaning of this below) the same set of guarded string as e:*



Let us now explain how to compute $G(s)$, the set of guarded strings accepted by a state s of an automaton $\mathcal{A}$ on guarded strings. A guarded string x is accepted by $\mathcal{A}$ if $x \in G(e)$ for some $e \in L(s)$, where $L(s) \subseteq (\mathtt{P} \cup \mathtt{B})^*$ is just the language accepted by s, as defined classically for non-deterministic automata and explained coalgebraically in Section 2.2. In the example above, we have $L(s) = \{\mathtt{b}_1, \mathtt{b}_2\mathtt{p}\}$ and thus

$$G(s) = G(\mathtt{b}_1) \cup G(\mathtt{b}_2\mathtt{p}) = G(\mathtt{b}_1 + \mathtt{b}_2\mathtt{p}).$$

Later in [14], Kozen showed that the deterministic version of automata on guarded strings (already defined in [12]) fits neatly in the coalgebraic framework: two expressions are bisimilar if and only if they recognize the same set of guarded strings.

A deterministic automaton on guarded strings is a pair $(S, \langle o_S, t_S\rangle)$ where $o\colon S \to \mathcal{B}$ (recall that $\mathcal{B}$ is the free Boolean algebra on $\mathtt{B}$, satisfying $\mathcal{B} \cong 2^{\mathtt{At}}$) and $t\colon S \to S^{\mathtt{At}\times\mathtt{P}}$. Note that formally a deterministic automaton on guarded strings is a *Moore automaton* (see Section 2.2).

We can obtain a deterministic automaton by using the following generalization of Brzozowski derivatives for KAT expressions (modulo ACI, as for classical regular expressions).

Definition 3 (Brzozowski derivatives for KAT expressions) *Given an expression $e \in \mathtt{Exp}$, we define $E\colon \mathtt{Exp} \to \mathcal{B} \cong 2^{\mathtt{At}}$ and $D\colon \mathtt{Exp} \to \mathtt{Exp}^{\mathtt{At}\times\mathtt{P}}$ by induction on the structure of e. First, $E(e)$ is given by:*

$$E(\mathtt{p}) = \emptyset \qquad E(\mathtt{b}) = \{\alpha \in \mathtt{At} \mid \alpha \leq b\} \qquad E(ef) = E(e) \cap E(f)$$
$$E(e+f) = E(e) \cup E(f) \qquad E(e^*) = \mathtt{At}$$

Next, we define $e_{\alpha\mathtt{q}} = D(e)(\langle \alpha, \mathtt{q}\rangle)$ by

$$\mathtt{p}_{\alpha\mathtt{q}} = \begin{cases} 1 & \text{if } \mathtt{p} = \mathtt{q} \\ 0 & \text{if } \mathtt{p} \neq \mathtt{q} \end{cases} \qquad \mathtt{b}_{\alpha\mathtt{q}} = 0 \quad (ef)_{\alpha\mathtt{q}} = \begin{cases} e_{\alpha\mathtt{q}}f + f_{\alpha\mathtt{q}} & \text{if } \alpha \in E(e) \\ e_{\alpha\mathtt{q}}f & \text{if } \alpha \notin E(e) \end{cases}$$

$$(e+f)_{\alpha\mathtt{q}} = e_{\alpha\mathtt{q}} + f_{\alpha\mathtt{q}} \qquad (e^*)_{\alpha\mathtt{q}} = e_{\alpha\mathtt{q}}e^*$$

The functions $\langle E, D\rangle$ provide $\mathtt{Exp}$ with a deterministic (Moore) automata structure, which leads, by finality (Theorem 3), to the existence of a unique homomorphism

$$\begin{tikzcd}
\mathrm{Exp} \arrow[r, dashed, "G"] \arrow[d, "\langle E,D\rangle"'] & (2^{\mathtt{At}})^{(\mathtt{At}\times\mathtt{P})^*} \cong 2^{\mathtt{GS}} \arrow[d, "\langle o_{\mathtt{GS}}, t_{\mathtt{GS}}\rangle"] \\
2^{\mathtt{At}} \times \mathrm{Exp}^{\mathtt{At}\times\mathtt{P}} \arrow[r] & 2^{\mathtt{At}} \times (2^{\mathtt{GS}})^{\mathtt{At}\times\mathtt{P}}
\end{tikzcd}$$

which assigns to each expression the language of guarded strings that it denotes. The coalgebra structure on $2^{\mathtt{GS}}$ is an instantiation of $\langle o_M, t_M\rangle$ as presented before Theorem 3 for the output set $B = 2^{\mathtt{At}}$ and input set $A = \mathtt{At} \times \mathtt{P}$. More precisely, we have

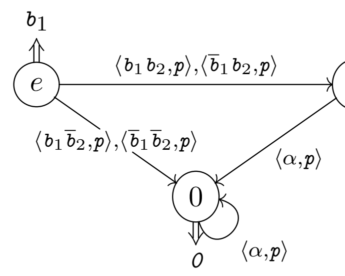
$$\begin{array}{lcl} o_{\mathtt{GS}}(f \in (2^{\mathtt{At}})^{(\mathtt{At}\times\mathtt{P})^*}) & = & f(\varepsilon) \\ t_{\mathtt{GS}}(f)(\langle \alpha, \mathtt{p}\rangle)(w) & = & f(\langle \alpha, \mathtt{p}\rangle w) \end{array}$$

or, alternatively, and equivalently,

$$\begin{array}{rcl} o_{\mathtt{GS}}(L \in 2^{\mathtt{GS}}) & = & \{\alpha \in \mathtt{At} \mid \alpha \in L\} \\ t_{\mathtt{GS}}(L)(\langle \alpha, \mathtt{p}\rangle) & = & \{w \in (\mathtt{At} \times \mathtt{P})^*) \mid \langle \alpha, \mathtt{p}\rangle w \in L\} \end{array}$$

The concrete definition of G, which can be deduced from the commutativity of the diagram above, is precisely the definition which appeared in the original paper on guarded strings and which we recalled in Definition 2.

Example 4 *The deterministic automaton of* $e = b_1 + b_2 p$*, which is the deterministic counterpart of the automaton in Example 3, would be*



b_1
1
e
1
0
$\langle b_1 b_2, p\rangle, \langle \overline{b}_1 b_2, p\rangle$
$\langle b_1 \overline{b}_2, p\rangle, \langle \overline{b}_1 \overline{b}_2, p\rangle$
$\langle \alpha, p\rangle$
$\langle \alpha, p\rangle$
0

since, for $B = \{b_1, b_2\}$, $At = \{b_1 b_2, b_1\overline{b}_2, \overline{b}_1 b_2, \overline{b}_1\overline{b}_2\}$ *and*

$$\begin{array}{ll} e_{b_1 b_2, p} = 0 + (b_2 p)_{b_1 b_2, p} = p_{b_1 b_2, p} = 1 & e_{\overline{b}_1 b_2, p} = 0 + (b_2 p)_{\overline{b}_1 b_2, p} = p_{\overline{b}_1 b_2, p} = 1 \\ e_{b_1 \overline{b}_2, p} = 0 + (b_2 p)_{b_1 \overline{b}_2, p} = 0 & e_{\overline{b}_1 \overline{b}_2, p} = 0 \end{array}$$

$$E(e) = \{\alpha \mid \alpha \leq b_1\} = \{b_1 b_2, b_1 \overline{b}_2\} \qquad E(0) = \emptyset \qquad E(1) = At$$

Above we represent the output $o_S(s)$ *of a state by* $\Rightarrow b$ *where* $b \in \mathcal{B}$ *is the element corresponding to the set* $o_S(s)$ *coming from the isomorphism* $2^{At} \cong \mathcal{B}$.

5 The Berry-Sethi Construction for KAT Expressions

In short, there are two types of automata recognizing regular sets of guarded strings:

$$S \to 2 \times (\mathcal{P}_\omega(S))^{\mathtt{P} \cup \mathtt{B}} \qquad\qquad S \to \mathcal{B} \times S^{\mathtt{At} \times \mathtt{P}}$$

The non-deterministic version has the advantage that it is very close to the expression, that is, one can easily compute the automaton from a given KAT expression and back, but its semantics is not coalgebraic. The deterministic version fits neatly into the coalgebraic framework, but it has the disadvantage that constructing the automaton from an expression inherits the same problems as in the Brzozowski construction: the number of equivalences that need to be decided increases exponentially. We propose here yet another type of automaton to recognize guarded strings: the construction from an expression to an automaton will be inspired by the Berry-Sethi construction presented in Section 3 and it is linear in the size of the expression.

Note that since KAT expressions can be interpreted as regular expressions over the extended alphabet $\mathtt{B} \cup \mathtt{P}$, the Berry-Sethi construction could be applied directly.

Theorem 6 *Let e be a KAT expression and $\mathcal{A}_{pos}(e)$ be the corresponding position automaton. Then, $G(e) = G(\mathcal{A}_{pos}(e))$.*

Proof: We know that $L(\mathcal{A}_{pos}(e)) = L(e)$. Now the result follows by using Kozen's observation in [12] that given a guarded string e and an automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L(e)$, one has $G(e) = G(\mathcal{A})$. □

The resulting automaton would have precisely the same type as the non-deterministic version of automata on guarded strings. However, there would be one state for each input symbol in $\mathtt{P} \cup \mathtt{B}$.

Example 5 *As an example take the expression $bp+\overline{c}$, whose marked version is $b_1p_2 + \overline{c}_3$. The resulting position automaton will have 4 states and the transition function will be given by the following table*

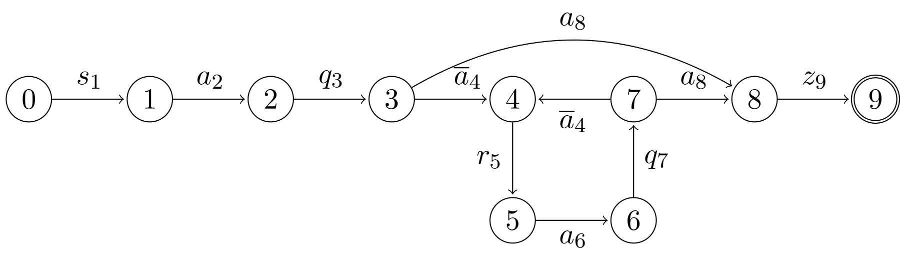
$follow$	
0	$\{1,3\}$
1	$\{2\}$
2	$\{!\}$
3	$\{!\}$

which results in the following automaton:

$$\begin{tikzcd}
0 \arrow[r, "b_1"] \arrow[d, "\overline{c}_3"'] & 1 \arrow[d, "p_2"] \\
3 & 2
\end{tikzcd}$$

Let us now present a more involved example. Consider the expression $saq(\overline{a}raq)^*az$*. The automaton will have* 10 *states, one for each symbol in the expression.*

$follow$	
0	$\{1\}$
1	$\{2\}$
2	$\{3\}$
3	$\{4, 8\}$
4	$\{5\}$
5	$\{6\}$
6	$\{7\}$
7	$\{8, 4\}$
8	$\{9\}$
9	$\{!\}$



As this last example shows, if we just apply the Berry-Sethi algorithm directly to a `KAT` expression, without distinguishing between tests and actions, the number of states increases very fast.

The construction we will show next includes only states for each atomic action in `P`, yielding smaller automata. From a given `KAT` expression e, we will construct an automaton (S, t) where $t\colon S \to \mathcal{B} \times \mathcal{P}_\omega(S)^{\mathcal{B}\times\mathtt{P}}$. This automaton type can be regarded as a compromise between the non-deterministic and deterministic versions of Kozen's automata.

We will start with generalizing the sets *first*, *follow* and *last*.

$$\begin{aligned} \mathit{first}(e) &= \{\langle \mathtt{b}, \mathtt{p}\rangle \mid \mathtt{b}_1\mathtt{b}_2\ldots\mathtt{b}_n\mathtt{p}x \in L(e)\,,\, \mathtt{b} = \bigvee(\mathtt{b}_1 \wedge \mathtt{b}_2 \wedge \ldots \mathtt{b}_n)\} \\ \mathit{follow}(e, \mathtt{p}) &= \{\langle \mathtt{b}, \mathtt{q}\rangle \mid x\mathtt{p}\mathtt{b}_1\mathtt{b}_2\ldots\mathtt{b}_n\mathtt{p}\mathtt{q}y \in L(e)\,,\, \mathtt{b} = \bigvee(\mathtt{b}_1 \wedge \mathtt{b}_2 \wedge \ldots \mathtt{b}_n)\} \\ \mathit{last}(e) &= \{\langle \mathtt{b}, \mathtt{p}\rangle \mid x\mathtt{p}\mathtt{b}_1\mathtt{b}_2\ldots\mathtt{b}_n \in L(e)\,,\, \mathtt{b} = \bigvee(\mathtt{b}_1 \wedge \mathtt{b}_2 \wedge \ldots \mathtt{b}_n)\} \end{aligned}$$

Note that the empty disjunction is `1` (and the empty conjunction is `0`). Below, we will use expressions of the form $e!$, where ! is a special end-marker, to avoid the computation of the last symbols that can be generated in e: $\langle \mathtt{b}, \mathtt{p}\rangle \in \mathit{last}(e) \Leftrightarrow \langle \mathtt{b}, !\rangle \in \mathit{follow}(e!, \mathtt{p})$.

Given a KAT expression e with all action symbols distinct we construct the automaton $\mathcal{A}_e = (\mathit{Pos}(e) \cup \{0\}, \langle o_S, t_S\rangle)$ where $\mathit{Pos}(e)$ is the number of distinct action symbols in e and

$$o_S(i) = \begin{cases} E(e) & \text{if } i = 0 \\ b & \text{if } i > 0 \text{ and } \langle b, !\rangle \in \mathit{follow}(e!, p_i) \\ \mathtt{0} & \text{otherwise} \end{cases}$$

and t is given by the following rules

$$\begin{tikzcd} 0 \arrow[r, "\langle b,p_j\rangle"] & j \end{tikzcd} \qquad \text{iff} \qquad \langle b, p_j\rangle \in \mathit{first}(e)$$

$$\begin{tikzcd} i \arrow[r, "\langle b,p_j\rangle"] & j \end{tikzcd} \qquad \text{iff} \qquad \langle b, p_j\rangle \in \mathit{follow}(e!, p_i)$$

The way the automaton is defined, state i will only have incoming transitions labeled by $\langle b, p_i\rangle$. Moreover, the fact that e has distinct symbols implies that the constructed automaton is deterministic, that is, $t\colon S \to \mathcal{B} \times S^{\mathcal{B}\times \mathtt{P}}$. Only after unmarking the labels $\mathtt{p}_i$ non-determinism will be introduced, as we will observe in an example below.

The guarded strings recognized by a state $s \in S$ of the automaton (S, t) where $t\colon S \to \mathcal{B} \times \mathcal{P}_\omega(S)^{\mathcal{B}\times \mathtt{P}}$ are now defined by the following rule

$$\begin{array}{lll} x \in G(s) & \Leftrightarrow & x = \alpha \text{ with } \alpha \leq E(s) \\ & \text{or} & x = \alpha \mathtt{p} x' \text{ with } x' \in G(s') \text{ for some } s' \in t_S(s)(\langle \mathtt{b}, \mathtt{p}\rangle) \\ & & \quad \text{and for some } \mathtt{b} \text{ s.t.} \alpha \leq \mathtt{b} \end{array}$$

Theorem 7 *Let e be a guarded string, with all action symbols distinct, and let $\mathcal{A}_e = (\mathit{Pos}(e) \cup \{0\}, \langle o_S, t_S\rangle)$ be the corresponding automaton constructed as above. Then, $G(e) = G(0)$.*

Proof: By induction on the structure of e.

If $e = \mathtt{b}$ then $GS(\mathtt{b}) = \{\alpha \mid \alpha \leq \mathtt{b}\}$ and $\mathcal{A}_e$ is a one state automaton with no transitions. Thus, $G(0) = \{\alpha \mid \alpha \leq E(\mathtt{b})\} = \{\alpha \mid \alpha \leq \mathtt{b}\} = G(b)$.

If $e = \mathtt{p}$ then $GS(\mathtt{p}) = \{\alpha \mathtt{p} \beta \mid \alpha, \beta \in \mathtt{At}\}$ and $\mathcal{A}_e$ is a two state automaton with only one transition from state 0 (with output $E(\mathtt{p}) = \mathtt{0}$) to state 1 (with output $\mathtt{1}$) labeled by $\langle \mathtt{1}, \mathtt{p}\rangle$. Thus,

$$G(0) = \{\alpha \mid \alpha \leq E(\mathtt{p})\} \cup \{\alpha \mathtt{p} \beta \mid \alpha \leq \mathtt{1}, \beta \leq \mathtt{1}\} = \{\alpha \mathtt{p} \beta \mid \alpha, \beta \in \mathtt{At}\} = G(\mathtt{p})$$

For $e = e_1 + e_2$, we have

$$
\begin{aligned}
& G(0) \\
= \ & \{\alpha \mid \alpha \leq E(e_1 + e_2)\} \cup \{\alpha \mathtt{p} x' \mid x' \in G(t_S(0)(\langle \mathtt{b}, \mathtt{p}\rangle)), \alpha \leq \mathtt{b}\} \\
= \ & \{\alpha \mid \alpha \leq E(e_1)\} \cup \{\alpha \mid \alpha \leq E(e_1)\} \ \cup \\
& \{\alpha \mathtt{p} x' \mid x' \in t_S(0)(\langle \mathtt{b}, \mathtt{p}\rangle), \alpha \leq \mathtt{b}, \mathtt{b}_1\mathtt{b}_2 \ldots \mathtt{b}_n \mathtt{p} x \in L(e_1 + e_2)\,, \\
& \qquad \mathtt{b} = \bigvee(\mathtt{b}_1 \wedge \mathtt{b}_2 \wedge \ldots \mathtt{b}_n)\} \\
= \ & \{\alpha \mid \alpha \leq E(e_1)\} \cup \{\alpha \mid \alpha \leq E(e_1)\} \ \cup \\
& \{\alpha \mathtt{p} x' \mid x' \in t_S(0)(\langle \mathtt{b}, \mathtt{p}\rangle), \alpha \leq \mathtt{b}, \mathtt{b}_1\mathtt{b}_2 \ldots \mathtt{b}_n \mathtt{p} x \in L(e_1)\,, \\
& \qquad \mathtt{b} = \bigvee(\mathtt{b}_1 \wedge \mathtt{b}_2 \wedge \ldots \mathtt{b}_n)\} \ \cup \\
& \{\alpha \mathtt{p} x' \mid x' \in t_S(0)(\langle \mathtt{b}, \mathtt{p}\rangle), \alpha \leq \mathtt{b}, \mathtt{b}_1\mathtt{b}_2 \ldots \mathtt{b}_n \mathtt{p} x \in L(e_2)\,, \\
& \qquad \mathtt{b} = \bigvee(\mathtt{b}_1 \wedge \mathtt{b}_2 \wedge \ldots \mathtt{b}_n)\} \\
\overset{IH}{=} \ & G(e_1) \cup G(e_2) \\
= \ & G(e_1 + e_2)
\end{aligned}
$$

Note that $\mathtt{b}_1\mathtt{b}_2 \ldots \mathtt{b}_n \mathtt{p} x \in L(e_i)$, for $i = 1, 2$, if and only if the state 0 of the automaton $\mathcal{A}_{e_i}$ has a transition labeled by $\langle \mathtt{b}, \mathtt{p}\rangle$ into some state.

For $e = e_1 e_2$, things get slightly more complicated. Let us start with the easy bit:

$$\alpha \in G(0) \Leftrightarrow \alpha \leq E(e_1 e_2) \Leftrightarrow \alpha \leq E(e_1) \text{ and } \alpha \leq E(e_2) \Leftrightarrow \alpha \in G(e_1 e_2)$$

Now take $\alpha_1 \mathtt{p}_1 \ldots \mathtt{p}_{n-1} \alpha_n \in G(0)$. This means that there exists a sequence of transitions:

$$\begin{tikzcd}
0 \arrow[r, "{\langle \mathtt{b}_1, \mathtt{p}_1\rangle}"] & \bullet \arrow[r, "{\langle \mathtt{b}_2, \mathtt{p}_2\rangle}"] & \ldots \arrow[r, "{\langle \mathtt{b}_{n-1}, \mathtt{p}_{n-1}\rangle}"] & \bullet \arrow[d, Rightarrow] \\
 & & & \mathtt{b}_n
\end{tikzcd}$$

such that $\alpha_i \leq b_i$, for all $i = 1, \ldots, n$. Because all the symbols in $e_1 e_2$ are distinct we can divide the above sequence of transitions as follows. Let p_k be the last action symbol in belonging to e_1. We have

$$\begin{tikzcd}
0 \arrow[r, "{\langle \mathtt{b}_1, \mathtt{p}_1\rangle}"] & \bullet \arrow[r, "{\langle \mathtt{b}_2, \mathtt{p}_2\rangle}"] & \ldots \arrow[r, "{\langle \mathtt{b}_k, \mathtt{p}_k\rangle}"] & \bullet \arrow[dlll, dashed, "{\langle \mathtt{b}_{k+1}, \mathtt{p}_{k+1}\rangle}"'] \\
\bullet \arrow[r, "{\langle \mathtt{b}_{k+2}, \mathtt{p}_{k+2}\rangle}"'] & \bullet \arrow[r, "{\langle \mathtt{b}_{k+3}, \mathtt{p}_{k+3}\rangle}"'] & \ldots \arrow[r, "{\langle \mathtt{b}_{n-1}, \mathtt{p}_{n-1}\rangle}"'] & \bullet \arrow[d, Rightarrow] \\
 & & & \mathtt{b}_n
\end{tikzcd}$$

and we observe that $\mathtt{b}_{k+1}$ is such that $x\mathtt{p}_k\mathtt{b}_{k+1}\mathtt{p}_{k+1}y \in L(e_1e_2)$. Thus, $\mathtt{b}_{k+1} = \mathtt{b}^1_{k+1}\mathtt{b}^2_{k+1}$ such that $x\mathtt{p}_k\mathtt{b}^1_{k+1} \in L(e_1)$ and $\mathtt{b}^2_{k+1}\mathtt{p}_{k+1}y \in L(e_2)$ and, as a consequence,

$$\langle b^1_{k+1}, p_k\rangle \in \mathit{last}(e_1) \text{ and } \langle b^2_{k+1}, p_{k+1}\rangle \in \mathit{first}(e_2)$$

Now we can conclude using the induction hypothesis since $\alpha_1\mathtt{p}_1 \dots \alpha_k\mathtt{p}_k\alpha_{k+1} \in G(0_1)$, where 0_1 is the state 0 of $\mathcal{A}_{e_1}$, and $\alpha_{k+1}\mathtt{p}_{k+1}\dots\alpha_{n-1}\mathtt{p}_{n-1}\alpha_n \in G(0_2)$, where 0_2 denotes the state 0 of $\mathcal{A}_{e_2}$, and therefore:

$$\begin{array}{rl} & \alpha_1\mathtt{p}_1 \dots \alpha_k\mathtt{p}_k\alpha_{k+1} \in G(e_1) \text{ and } \alpha_{k+1}\mathtt{p}_{k+1}\dots\alpha_{n-1}\mathtt{p}_{n-1}\alpha_n \in G(e_2) \\ \Leftrightarrow & \alpha_1\mathtt{p}_1 \dots \alpha_k\mathtt{p}_k\alpha_{k+1}\mathtt{p}_{k+1}\dots\alpha_{n-1}\mathtt{p}_{n-1}\alpha_n \in G(e_1e_2) \end{array}$$

The case e^* follows a similar reasoning as in e_1e_2 and is left to the reader. □

This theorem refers to marked expressions. Note, however, that unmarking the labels of the automaton only changes the action symbols and it will also yield $G(\overline{0}) = \overline{G(0)}$, where $G(\overline{0})$ denotes the set of guarded strings recognized by state 0 of the unmarked automaton and $\overline{G(0)}$ the unmarking of the set of guarded strings recognized by state 0 of the marked automaton.

Next, we present an algorithm to compute the sets *first*, *follow* and *last* for `KAT` expressions.

Proposition 2 *Let e be a* `KAT` *expression with distinct symbols.* $\mathbf{F}$, *defined by the rules below, is such that* $\mathbf{F}(e, \{\langle 1, !\rangle\})$ *yields a set of pairs of the form* $\langle p_i, \mathit{follow}(e!, p_i)\rangle$. *The rules are:*

$$\begin{array}{lcl} \mathbf{F}(e_1 + e_2, S) & = & \mathbf{F}(e_1, S) \cup \mathbf{F}(e_2, S) \\ \mathbf{F}(e_1.e_2, S) & = & \mathbf{F}(e_1, \mathit{first}(e_2) \cup E(e_2).S) \cup \mathbf{F}(e_2, S) \\ \mathbf{F}(e_1^*, S) & = & \mathbf{F}(e_1, \mathit{first}(e_1) \cup S) \\ \mathbf{F}(p, S) & = & \{\langle p, S\rangle\} \\ \mathbf{F}(b, S) & = & \emptyset \end{array}$$

where

$$\begin{array}{lcl} \mathit{first}(e_1 + e_2) & = & \mathit{first}(e_1) \cup \mathit{first}(e_2) \\ \mathit{first}(e_1.e_2) & = & \mathit{first}(e_1) \cup E(e_1).\mathit{first}(e_2) \\ \mathit{first}(e_1^*) & = & \mathit{first}(e_1) \\ \mathit{first}(p) & = & \{\langle 1, p\rangle\} \\ \mathit{first}(b) & = & \emptyset \end{array}$$

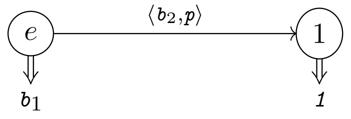
Note the similarities between Propositions 2 and 1 (the proofs are also similar and hence we do not include them here). The fact that the Boolean algebra $\mathcal{B}$ generalizes the two element Boolean algebra of classical regular expressions is reflected in the clause for the concatenation in the following way. The test for empty word $o_{\mathcal{R}}$ is replaced by the Boolean value of a KAT expression e and the multiplication is now redefined to propagate the tests:

$$\mathtt{b}.S = \begin{cases} \emptyset & \text{if } \mathtt{b} = \mathtt{0} \\ \{\langle \mathtt{bb}', \mathtt{p}\rangle \mid \langle \mathtt{b}', \mathtt{p}\rangle \in S\} & \text{otherwise} \end{cases}$$

Example 6 *We show now two examples of the algorithm above. We start with applying to the expression* $e = b_1 + b_2p$, *which we already used in Examples 3 and 4. This expression already has all action symbols distinct so no marking is needed. First, we compute* $\mathbf{F}(e, \{\langle 1, !\rangle\})$*:*

$$\begin{aligned}\mathbf{F}(b_1 + b_2p, \{\langle 1, !\rangle\}) &= \mathbf{F}(b_1, \{\langle 1, !\rangle\}) \cup \mathbf{F}(b_2p, \{\langle 1, !\rangle\}) \\ &= \mathbf{F}(p, \{\langle 1, !\rangle\}) \\ &= \{\langle p, \{\langle 1, !\rangle\}\rangle\}\end{aligned}$$

Thus, because $\mathit{first}(e) = \{\langle b_2, p\rangle\}$ *and* $E(e) = b_1$, *then* $\mathcal{A}_e$ *is given by*

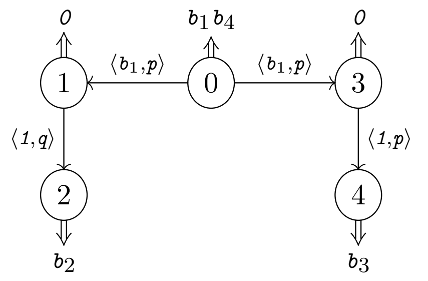


Next, we consider the expression $e_1 = b_1(pqb_2 + ppb_3 + b_4)$. *We have* $E(e_1) = b_1b_4$, $\overline{e_1} = b_1(p_1q_2b_2 + p_3p_4b_3 + b_4)$ *and*

$$\begin{aligned} & \mathit{first}(\overline{e_1}) \\ = \;& \mathit{first}(b_1) \cup E(b_1).\mathit{first}((p_1q_2b_2 + p_3p_4b_3 + b_4)) \\ = \;& b_1.\{\langle 1, p_1\rangle, \langle 1, p_3\rangle\} = \{\langle b_1, p_1\rangle, \langle b_1, p_3\rangle\} \end{aligned}$$

$$\begin{aligned} & \mathbf{F}(\overline{e_1}, \{\langle 1, !\rangle\}) \\ = \;& \mathbf{F}(p_1q_2b_2 + p_3p_4b_3 + b_4, \{\langle 1, !\rangle\}) \\ = \;& \mathbf{F}(p_1q_2b_2, \{\langle 1, !\rangle\}) \cup \mathbf{F}(p_3p_4b_3, \{\langle 1, !\rangle\}) \\ = \;& \mathbf{F}(p_1, \{\langle q_2, 1\rangle\}) \cup \mathbf{F}(q_2, E(b_2).\{\langle 1, !\rangle\}) \cup \mathbf{F}(p_3, \{\langle 1, p_4\rangle\}) \\ & \cup \mathbf{F}(p_4, E(b_3).\{\langle 1, !\rangle\}) \\ = \;& \{\langle p_1, \{\langle 1, q_2\rangle\}\rangle, \langle q_2, \{\langle b_2, !\rangle\}\rangle, \langle p_3, \{\langle 1, p_4\rangle\}\rangle, \langle p_4, \{\langle b_3, !\rangle\}\rangle\} \end{aligned}$$

The automaton $\mathcal{A}_{e_1}$, after unmarking, is then given by:



The non-deterministic version of Kozen's automata on guarded strings would have 7 *states and* 8 *transitions, whereas the (minimal) deterministic version would have* 5 *states (same as the automaton above), but* $8 \times 8 = 64$ *transitions since for* $\mathtt{B} = \{b_1, b_2, b_3\}$ *the set* **At** *has* 8 *elements.*

6 Conclusion

This paper contains an exercise on KAT expressions. First, we show that one can compile a non-deterministic automaton (on guarded strings) from a KAT expression by directly applying the Berry-Sethi construction, a very efficient algorithm for classical regular expressions. Secondly, we present a new automata model for KAT expressions, which can be seen as a compromise between Kozen's deterministic and non-deterministic models. We then adapt the Berry-Sethi construction to the new model. Compiling KAT expressions into the new model will yield automata with fewer states, which is an important feature for certain applications on program verification. The constructed automata have however more transitions and it remains to be explored how in practice this affects the efficiency of the construction.

References

[1] Allegra Angus and Dexter Kozen. Kleene algebra with tests and program schematology. Technical Report TR2001-1844, Computing and Information Science, Cornell University, July 2001.

[2] Valentin M. Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theor. Comput. Sci.*, 155(2):291–319, 1996. doi:10.1016/0304-3975(95)00182-4.

[3] Falk Bartels. *On generalized coinduction and probabilistic specification formats*. PhD thesis, Vrije Universiteit Amsterdam, 2004. PhD thesis.

[4] Gérard Berry and Ravi Sethi. From regular expressions to deterministic automata. *Theor. Comput. Sci.*, 48(3):117–126, 1986. doi:10.1016/0304-3975(86)90088-5.

[5] Janusz A. Brzozowski. Derivatives of regular expressions. *Journal of the ACM*, 11(4):481–494, 1964. doi:10.1145/321239.321249.

[6] Jean-Marc Champarnaud, Florent Nicart, and Djelloul Ziadi. From the ZPC structure of a regular expression to its follow automaton. *IJAC*, 16(1):17–34, 2006. doi:10.1142/S0218196706002895.

[7] Jean-Marc Champarnaud and Djelloul Ziadi. Canonical derivatives, partial derivatives and finite automaton constructions. *Theor. Comput. Sci.*, 289(1):137–163, 2002. doi:10.1016/S0304-3975(01)00267-5.

[8] V.M. Glushkov. The abstract theory of automata. *Russian Math. Surveys*, 16:1–53, 1961. doi:10.1070/RM1961v016n05ABEH004112.

[9] Bart Jacobs. A bialgebraic review of deterministic automata, regular expressions and languages. In Kokichi Futatsugi, Jean-Pierre Jouannaud, and José Meseguer, editors, *Essays Dedicated to Joseph A. Goguen*, volume 4060 of *Lecture Notes in Computer Science*, pages 375–404. Springer, 2006. doi:10.1007/11780274_20.

[10] Donald M. Kaplan. Regular expressions and the equivalence of programs. *J. Comput. Syst. Sci.*, 3(4):361–386, 1969. doi:10.1016/S0022-0000(69)80027-9.

[11] Dexter Kozen. *Automata and Computability*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.

[12] Dexter Kozen. Automata on guarded strings and applications. *Matemática Contemporânea*, 24:117–139, 2003.

[13] Dexter Kozen. Nonlocal flow of control and Kleene algebra with tests. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 105–117. IEEE Computer Society, 2008. doi:10.1109/LICS.2008.32.

[14] Dexter Kozen. On the coalgebraic theory of Kleene algebra with tests. Technical Report http://hdl.handle.net/1813/10173, Computing and Information Science, Cornell University, March 2008.

[15] Dexter Kozen and Maria-Christina Patron. Certification of compiler optimizations using Kleene algebra with tests. In John W. Lloyd, Verónica Dahl, Ulrich Furbach, Manfred Kerber, Kung-Kiu Lau, Catuscia Palamidessi, Luís Moniz Pereira, Yehoshua Sagiv, and Peter J. Stuckey, editors, *Computational Logic*, volume 1861 of *Lecture Notes in Computer Science*, pages 568–582. Springer, 2000. doi:10.1007/3-540-44957-4_38.

[16] Robert McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IRE Transactions on Electronic Computers*, 9(1):39–47, 1960. doi:10.1109/TEC.1960.5221603.

[17] Michael O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:114–125, 1959. doi:10.1147/rd.32.0114.

[18] Jan J. M. M. Rutten. Automata and coinduction (an exercise in coalgebra). In Davide Sangiorgi and Robert de Simone, editors, *CONCUR*, volume 1466 of *Lecture Notes in Computer Science*, pages 194–218. Springer, 1998. doi:10.1007/BFb0055624.

[19] Jan J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.*, 249(1):3–80, 2000. doi:10.1016/S0304-3975(00)00056-6.

[20] Alexandra Silva. *Kleene coalgebra.* PhD thesis, Radboud University Nijmegen, 2010.

[21] Ken Thompson. Programming techniques: Regular expression search algorithm. *Commun. ACM*, 11(6):419–422, 1968. doi:http://doi.acm.org/10.1145/363347.363387.