Formal Semantics for Probabilistic Verification of Stochastic Regular Expressions

Sinem Getir[1], Esteban Pavese[1], and Lars Grunske[1]

Software Engineering, Humboldt University Berlin, Germany

Abstract. Modelling and verification of software systems is an effective phase of system development, as it can uncover failures in design early in the development process. There is an increasing need for languages and processes that allow for the specification of *uncertainty* that allow, for example, the modelling of the unknown behaviour of a user, or the stochastic failure rate of hardware components.
In this paper we introduce a formal semantics on Stochastic Regular Expressions (SREs) over probabilistic action logics for quantitative verification. We provide the recursive calculation of the language generated by an SRE, enhanced to reuse local results for global verification of system specifications. Furthermore, we demonstrate how to model systems with SREs and how to perform reachability analysis with Probabilistic Action-based Computational Tree Logic (PACTL*).

Keywords: stochastic regular expressions, probabilistic formalism, probabilistic model checking, probabilistic verification

1 Introduction

The analysis of systems with a probabilistic behaviour plays an important role in several applications, such as software engineering, speech recognition, digital communications and computational biology among others. On the other hand, regular expressions have spread through all of theoretical computer science and enjoy plentiful applications in the field of natural language processing, including parsing, deep language models, model inference and machine translation.

Several studies have been conducted in probabilistic version of regular expressions that is studied as probabilistic concurrent Kleene algebra in [12] and extended with additional Kleene theorems in the application of quantitative reasoning on database queries [3].

Kartzow and Weidner [7,17] define a Monadic Second-Order Logic (MSOL) and a constraint logic with temporal properties for data analysis for probabilistic regular expressions. Additionally, Weidner specified Probabilistic Regular Expressions for infinite strings with ω-properties in his thesis [16].

In this paper, we define a semantics of stochastic regular expressions in the context of probabilistic model checking, employing a probabilistic extension of Action-based Computation Tree Logic (ACTL*) to reason about temporal properties quantitatively. Probabilistic model checking [8] is a technique developed

in order to automatically perform such assessments, and has been successfully applied in recent years [10]. Probabilistic models used for model checking can be at different levels of abstraction, such as Markov Chains [2], Markov Decision Processes [2] and Stochastic Petri Nets [11] among others. In contrast to state-based representations, we introduce an approach and focus on stochastic regular expressions as an input model for probabilistic model checking applications.

2 Stochastic Regular Expressions (SRE)

In this section, we briefly recall SRE [14] and action logic [1].

2.1 Syntax of SRE

The syntax of a Stochastic Regular Expression (SRE) E over an alphabet Σ is defined recursively as follows:

$$E := \alpha \;\Big|\; \sum_i E_i[n_i] \;\Big|\; E_1 : E_2 \;\Big|\; E^{*f} \tag{1}$$

with $\alpha \in \Sigma \cup \{\varepsilon\}$, $n_i \in \mathbb{N}_0$, $f \in [0,1] \subset \mathbb{R}$ and every term E_i is a SRE, such that:

1. *Atomic Action* α: α is an atomic action that belongs to the alphabet Σ.
2. *Choice* $\sum_i E_i[n_i]$: One of the provided terms $E_i[n_i]$ is probabilistically chosen according to calculated probabilities from occurence values. n_i denotes the *occurrence value* or *choice rate* for each term, such that the i-th term is chosen with probability $\frac{n_i}{\sum_j n_j}$. *Occurrence value* or *choice rate* is defined as the number of cases that the node is chosen statistically.
3. *Concatenation* $E_1 : E_2$: The terms E_1 and E_2 are successively interpreted.
4. *Kleene Closure* E^{*f}: The term E is repeated for an indefinite number of times, subject to a binomial distribution. Each iteration occurs with a probability of f. The termination probability is $1 - f$.
5. *Plus Closure* E^{+f}: The *+Closure* is a syntactic sugar that is omitted here, but can be easily emulated with $E : E^{*f}$.

Without loss of generality, the empty string ε is not included in the alphabet. However we include the empty string ϵ as an atomic action. On the other hand, ϵ can be derived from an expression like $\alpha^{*0.0}, \alpha \in \Sigma$.

A derivation of a conventional regular expression E is the set of sentences, or strings over the alphabet, derivable from it. This defines the language L(E) of E. This notion of language derivation is similarly applicable to SRE, except that each string has a probability value associated with it, and hence the language itself is associated with a probability distribution of its members as explained in the following within its denotational semantics.

2.2 Denotational semantics of SREs

Intuitively SREs can be understood as an expression that defines a specific probability function over its language strings such that $[\![E]\!] : s \in \Sigma^* \rightarrow [0,1]$. Such probabilistic language (p-language) is previously described for discrete events systems in [9]. In the following, we provide a trivial example on the p-language to explain the relationship between the SRE and the p-language.

The semantics of SREs are described by the p-language [9] in the style of denotational semantics [15]. The probability function for an SRE E is denoted by $[\![E]\!]$, and its application to a particular string s is denoted $[\![E]\!]s$, which represents an acceptance probability associated with string s in the language L(E). The probability function recursively calculates the occurrence probability of an arbitrary string $s \in \Sigma^*$ in a SRE model. By definition, if an arbitrary string s has a probability value greater than 0, it is accepted as a word of SRE. Meaning that if $s \in L(E)$ then $[\![E]\!]s > 0$.

Example 1. Let L be a probabilistic language [9] describing the Bernoulli process where each experiment has two outcomes a and b with probabilities p and $1-p$ respectively. Then L is defined on the alphabet $\Sigma = \{a, b\}$ as $L(s) = p^{\#(a,s)}.(1-p)^{\#(b,s)}$ where $\#(a,s)$ is representing the number of occurrences in word s.

In the following we formally define some additional notions which will be referred to throughout this paper.

Definition 1 (Words of a SRE). $Words(E) = \{w \mid w \in L(E)\}$

Definition 2 (Word length). *A **length** of a word $w = w_0 w_1 ... w_n$ is the number of included characters and denoted as $|w| = n$ through the paper.*

Definition 3 (Probability of a word in the language). *The probability function $[\![E]\!]s$ for every possible SRE term E ($\alpha \mid \sum E_i(n_i) \mid E_1 : E_2 \mid E^{*f} \mid E^{+f}$) is calculated recursively, where $s = \alpha_1, ..\alpha_n \in \Sigma^*$:*

- ***Atomic actions:***

$$[\![\alpha]\!]s = 1, \textit{ if } \alpha = s$$
$$[\![\alpha]\!]s = 0, \textit{ if } \alpha \neq s \tag{2}$$

- ***Choice***

$$\Big[\!\Big[\sum E_i n_i \Big]\!\Big] s = \sum_k \Big(\frac{n_k}{\sum n_i} \Big) \cdot [\![E_k]\!]s \tag{3}$$

  Since every term might recognize s, the overall probability for a choice expression is the sum of all the term probabilities with respect to s.

- ***Concatenation**:*

$$[\![E_1 : E_2]\!]s = \sum_{i=0}^{n} ([\![E_1]\!]\alpha_1..\alpha_i \cdot [\![E_2]\!]\alpha_{i+1}..\alpha_n)$$

In the summation, s is decomposed into two (possibly empty) substrings, each of which may be consumed by a concatenated expression. Even though one term may recognize its substring argument, if the other term does not recognize its respective substring, then that term returns a probability of 0, and the overall probability for that instance of decomposition is 0.

– ***Kleene closure**:*

$$\begin{aligned} [\![E^{*f}]\!]\varepsilon &= 1 - f \\ [\![E^{*f}]\!]s &= \sum_{i=1}^{n} (f \cdot [\![E]\!]\alpha_1..\alpha_i \cdot [\![E^{*f}]\!]\alpha_{i+1}..\alpha_n) \\ &\quad + f \cdot [\![E]\!]s \cdot [\![E^{*f}]\!]\varepsilon \end{aligned} \tag{4}$$

The first formula accounts for empty strings, as the only way an iterated expression should recognize an empty string is by not iterating; in other words terminating without executing (The termination probability is therefore 1-f). The other formula recursively defines the general case. Here, one iteration of E will consume some portion of s, and the rest of s is consumed by further iterations. It is assumed that an iteration of a loop always consumes some non-empty string. Otherwise, the semantic model would have to account for Kleene closure iterating indefinitely on an argument, which is not an useful behaviour.

All SRE probability functions presented above are well formed probabilitity functions. Interested readers can find the details of probability functions and the proof of well formness in [14].

2.3 Action based Computation Tree Logic (ACTL*)

ACTL* is introduced in [4] where the comparison for state labelled and transition labelled systems is studied. The syntax of ACTL* is described recursively on action labelled transition systems as follows; where φ is a formula executed on the runs of the system:

$$\varphi := True \mid \neg\varphi \mid \varphi \wedge \varphi\prime \mid \exists\varphi \mid \varphi\, \mathcal{U}\, \varphi\prime \mid \mathcal{X}_a\varphi \mid \mathcal{X}\varphi \tag{5}$$

We biefly recall the definitions required for the ACTL* semantics.

Definition 4 (Labelled transition systems). *A labelled transition system is a tuple $(S, Act, \rightarrow)$ where:*

. *S is a finite set of states*
. *Act is a finite, non-empty set of actions*
. *$\rightarrow$ is the transition relation denoted in $\subseteq S \times (Act \cup \epsilon) \times S$ and any element of $\rightarrow$ is called a transition.*

Definition 5. *A sequence* $(s_0, \alpha_0, s_1)(s_1, \alpha_1, s_2).. \in \ \rightarrow \infty$ *is called a* ***path*** *from* s_0*. A* ***run*** $\rho = (s, \Phi)$ *is a pair from* $s \in S$*, where* Φ *is a path from* s*, first state of* ρ *is* s $(first(\rho) = s)$ *and* $path(\rho) = \Phi$*. If a* ***run*** θ *is a suffix for* ***run*** ρ*, then we denote as* $\theta \leq \rho$*.*

The semantics is given by satisfaction relations based on the definitions above:

$\rho \models True$ always
$\rho \models \neg\varphi$ iff $\rho \not\models \varphi$
$\rho \models \varphi \wedge \varphi'$ iff $\rho \models \varphi$ and $\rho \models \varphi'$
$\rho \models \exists\psi$ iff there exists a run $\theta \in run(first(\rho))$ such that $\theta \models \psi$
$\rho \models \psi \,\mathcal{U}\, \psi'$ iff there exists a θ with $\rho \leq \theta$ such that $\theta \models \psi\prime$ and for all $\rho \leq \eta \leq \theta : \eta \models \psi\prime$
$\rho \models \mathcal{X}\varphi$ iff there exists s, α, $s\prime$, θ such that $\rho = (s, (s, \alpha, s\prime))\theta$ and $\theta \models \varphi$
$\rho \models \mathcal{X}_a\varphi$ iff there exists s, $s\prime$, θ such that $\rho = (s, (s, a, s\prime))\theta$ and $\theta \models \varphi$ (6)

3 Semantics of Stochastic Regular Expression Trees with Probabilistic Action based Computation Tree Logic

Our goal is to reason about temporal properties on SRE models probabilistically. A very common logic Probabilistic computation tree logic (PCTL*) [6] and variants are defined on the state and path formulas. However Stochastic Regular Expressions do not have the explicit notation of a state. Therefore we prefer to extend the ACTL* logic semantically expressed on the **runs** of the system as provided in subsection 2.3.

The extended syntax of a Probabilistic ACTL* (PACTL*) is defined as follows where φ is a word and Φ is a SRE formula.

Definition 6 (Syntax of PACTL*).

$$\Phi = \neg\Phi \mid \Phi \wedge \Phi' \mid \mathcal{P}_P(\varphi) \tag{7}$$
$$\varphi = true \mid \mathcal{X}_a\varphi \mid \mathcal{X}\varphi \mid \varphi \,\mathcal{U}\, \varphi' \tag{8}$$

where $P \subseteq [0, 1]$

3.1 Semantics of PACTL*

Every SRE term E is defined as a node that specifies **operation type** (*choice, concatenation, kleene closure* or *action*)**, choice rate, kleene probability and subnode(s)**.

Formally E is an *action* or a tuple based on its type: $(\mathcal{N}, \mathcal{R})$, $(\mathcal{N})$, $(\mathcal{N}, k)$ if the opetaion types are choice, concatenation, kleene closure ($\mathcal{T} = + \mid : \mid *$ or action $a \in \Sigma$,) respectively. where $\mathcal{N} = \{E_1, E_2, ..., E_n\}$ is the finite set of subnodes in the operating order $(E_1 : E_2 \neq E_2 : E_1)$, $\mathcal{R} = \{r_1, r_2, ..r_n\}$ is the set of corresponding **choice rates** ($r \in \mathbb{N}$) and $k \in [0, 1]$ is a **kleene probability**.

We construct every SRE node in a bottom up fashion by parsing the given string by using operator presedence. Hence, we avoid the ambiguity of the parsed trees and remain them unique. (The order of operator predence is $*, :, +$). We also restrict one operation type per SRE node which allows unambiguity.

The semantics is defined a satisfaction relation for a word w and a SRE term E as follows:

$$E \models \neg\Phi \text{ iff } E \not\models \Phi \tag{9}$$
$$E \models \Phi \wedge \Phi' \text{ iff } E \models \Phi \wedge E \models \Phi' \tag{10}$$
$$E \models \mathcal{P}_P(\varphi) \text{ iff } Pr\{w \models \varphi \mid w \in Words(E)\} \in P \tag{11}$$
$$w \models true \text{ always} \tag{12}$$
$$w \models \mathcal{X}_a\varphi \text{ iff } w[0] = a \text{ and } w[1] \models \varphi \tag{13}$$
$$w \models \mathcal{X}\varphi \text{ iff } w[1] \models \varphi \tag{14}$$
$$w \models \varphi\,\mathcal{U}\,\varphi\prime \text{ iff for some } i \leq |w|,\ w[i] \models \varphi' \text{ and } w[j] \models \varphi,\ \forall j < i. \tag{15}$$

The set of words for an SRE can be calculated recursively on SRE node E:

$$Words(E) = \begin{cases} \{a\}, & \text{if } \mathcal{T} = a \\ \bigcup\limits_{E_i \in \mathcal{N}} Words(E_i), & \text{if } \mathcal{T} = + \\ (Words(E_0) \cdot Words(E_1)) \ldots \cdot Words(E_n), \forall E_i \in \mathcal{N} & \text{if } \mathcal{T} =: \\ \bigcup\limits_{k}^{\infty} Words(E_{\text{sub}})^k, \text{ where } E_{\text{sub}} \in \mathcal{N} \text{ and } E = (E_{\text{sub}})^* & \text{if } \mathcal{T} = * \end{cases} \tag{16}$$

A system is specified as an SRE tree that includes a root node and finite set of nodes. An SRE tree is formally defined as; $T_E = (E_{\text{root}}, \mathcal{E}, \Sigma)$, where E_{root} is the root node, $\mathcal{E}$ is the finite set of all nodes and Σ is the alphabet. Hence, verifying the the root node E_{root} will result in verifying the system.

4 Example: System Specification with SRE Tree

We provide an example automata and a corresponding SRE system specification in the following paragraphs. Let us assume that we have a system that is composed of some web services aiming to achieve a message protocol. The subcomponents Service $1(S_1)$ and Service $2(S_2)$ are executing the login of sytem and message sending and logging out from the system respectively. The system can be defined as a stochastic regular expression tree as follows:

$$
\begin{aligned}
&T_E = (E_{\text{root}}, \mathcal{E}, \Sigma)\\
&E_{\text{root}} = S\\
&\mathcal{E} = \{S, S_1, S_2, E_1, E_2, E_3, E_4, E_5, E_6\}\\
&\Sigma = \{start, login, authenticationFail, logout, sendMsg,\\
&msgFail, terminate, success, retry\}\\
&S = S_1 : S_2\\
&S_1 = start : login\\
&S_2 = authenticationFail[15] + logout[20] + E_1[65]\\
&E_1 = sendMsg : E_2\\
&E_2 = msgFail[5] + E_3[95]\\
&E_3 = E_4 : E_5\\
&E_4 = E_6^{*0.25}\\
&E_5 = logout : terminate\\
&E_6 = success : retry
\end{aligned}
$$

Let a PACTL* formula $\mathcal{P}_{[0.3,0.4]}\Big(true\,\mathcal{U}\,(\mathcal{X}_{\text{msgFail}})true\Big)$ for the analysis on T_E. The formula indicates the reachability analysis of the action "msgFail" on the root node $E_{\text{root}} = S$. The words reaching the "msgFail' from S are then recursively calculated:

$Words(S)_{[\text{reaching"msgFail"}]} \subset Words(S) = \{start.login.sendMsg.msgFail\ (0.0325),$
$start.login.sendMsg.succes.retry.msgFail\ (0.008125),$
$start.login.sendMsg.succes.retry.succes.retry.msgFail\ (0.00203125), ...\}$

The union is then

$$0.0325 \times \prod_{i=0}^{\infty}(0.25)^i = 0.0325 \in [0.3, 0.4] \tag{17}$$

$$S \models \mathcal{P}_{[0.3,0.4]}\Big(true\,\mathcal{U}\,(\mathcal{X}_{\text{msgFail}})true\Big) \tag{18}$$

Visually, we provide the corresponding probabilistic automata in Figure 1. The proof of equivalence between probabilistic Rabin automata [13] and the p-language, on which stochastic regular expression's semantics denoted, is provided in [5].

SREs are calculated in a bottom up way by remaining the probabilistic calculations of strings. Such techique enables to reach every calculation on each node locally. The idea is to calculate all information on every SRE term and compose the solutions based on the operations.

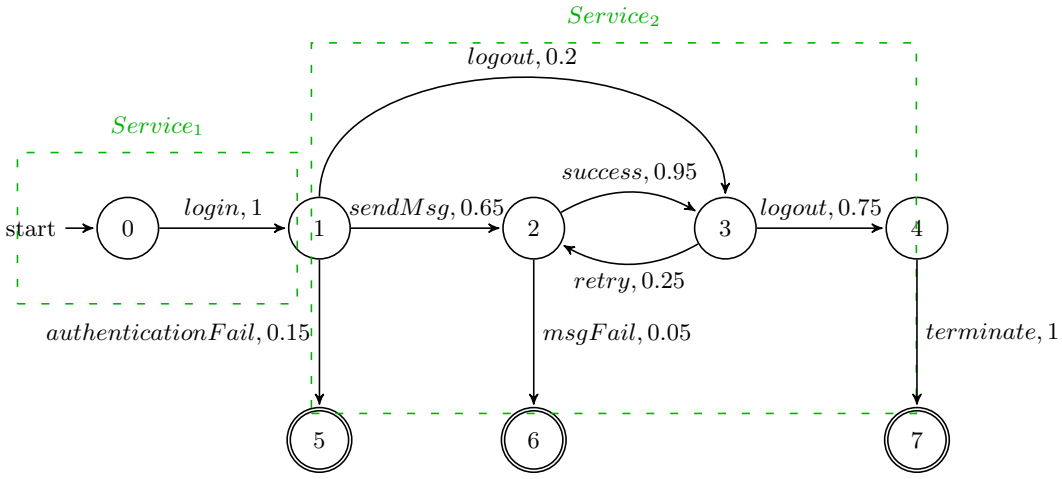

Fig. 1. Corresponding probabilistic automata to system T_E

5 Conclusion

We described a formal semantics for model checking of SREs that enjoys various applications in computer science. We studied the stochastic regular expressions with action based probabilistic logic in the model checking context and used stochastic regular expressions as an input model. Our initial attempt to reachability analysis with strings is also presented which is promising and convenient for parallel and incremental computation especially in the domain of component based systems or modular systems. The further investigation is to extend the reachibility analysis for the application of full PACTL* on SRE trees and evaluate our approach with set of models. Furthermore, we are planning to use SRE model checking for the incremental computation of local changes that can occur in the model.

References

1. Baier, C., Cloth, L., Haverkort, B.R., Kuntz, M., Siegle, M.: Model checking markov chains with actions and state labels. IEEE Trans. Software Eng. 33(4), 209–224 (2007)
2. Baier, C., Katoen, J.P.: Principles of Model Checking (Representation and Mind Series). The MIT Press (2008)
3. Bollig, B., Gastin, P., Monmege, B., Zeitoun, M.: A probabilistic kleene theorem. In: Chakraborty, S., Mukund, M. (eds.) Automated Technology for Verification and Analysis. pp. 400–415. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
4. De Nicola, R., Vaandrager, F.: Action versus state based logics for transition systems. In: Guessarian, I. (ed.) Semantics of Systems of Concurrent Processes. pp. 407–419. Springer Berlin Heidelberg, Berlin, Heidelberg (1990)

5. Garg, V.K., Kumar, R., Marcus, S.I.: A probabilistic language formalism for stochastic discrete-event systems. IEEE Transactions on Automatic Control 44(2), 280–293 (Feb 1999)
6. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing 6(5), 512–535 (Sep 1994)
7. Kartzow, A., Weidner, T.: Model checking constraint LTL over trees. CoRR abs/1504.06105 (2015)
8. Katoen, J.P.: The probabilistic model checking landscape. In: Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science. pp. 31–45. LICS '16, ACM, New York, NY, USA (2016)
9. Kumar, R., Garg, V.K.: Control of stochastic discrete event systems modeled by probabilistic languages. IEEE Trans. Automat. Contr. 46(4), 593–606 (2001)
10. Kwiatkowska, M., Norman, G., Parker, D.: Probabilistic model checking in practice: Case studies with PRISM. ACM SIGMETRICS Performance Evaluation Review 32(4), 16–21 (2005)
11. Marsan, M.A.: Stochastic petri nets: An elementary introduction. In: Rozenberg, G. (ed.) Advances in Petri Nets 1989. pp. 1–29. Springer Berlin Heidelberg, Berlin, Heidelberg (1990)
12. McIver, A., Rabehaja, T.M., Struth, G.: Probabilistic concurrent kleene algebra. In: Proceedings 11th International Workshop on Quantitative Aspects of Programming Languages and Systems, QAPL 2013, Rome, Italy, March 23-24, 2013. pp. 97–115 (2013)
13. Rabin, M.O.: Probabilistic automata. Information and Control 6(3), 230–245 (1963)
14. Ross, B.J.: Probabilistic pattern matching and the evolution of stochastic regular expressions. Applied Intelligence 13(3), 285–300 (Nov 2000)
15. Stoy, J.E.: Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory. MIT Press, Cambridge, MA, USA (1977)
16. Weidner, T.: Probabilistic Logic, Probabilistic Regular Expressions, and Constraint Temporal Logic. PhD dissertation, University Leipzig (2012)
17. Weidner, T.: Probabilistic Regular Expressions and MSO Logic on Finite Trees. In: Harsha, P., Ramalingam, G. (eds.) 35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015). Leibniz International Proceedings in Informatics (LIPIcs), vol. 45, pp. 503–516 (2015)