BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2017
http://excel.fit.vutbr.cz

Comparing Languages and Reducing Automata Used in Network Traffic Filtering

Vojtěch Havlena*

Abstract

The focus of this paper is the comparison of languages and reduction of automata used in network traffic monitoring. We propose several approaches for approximate (language non-preserving) reduction of automata and comparison of their languages. The reductions are based on either under-approximating the languages of automata by pruning their states, or over-approximating the language by introducing new self-loops (and pruning redundant states later). Our approximate reduction methods and the proposed probabilistic distance utilize information from a network traffic. We provide formal guarantees with respect to a model of network traffic, represented using a probabilistic automaton. We implemented the methods and evaluated them on automata used in network traffic filtering.

Keywords: Automata reduction — Language distance — Finite automaton — Network traffic monitoring

Supplementary Material: Downloadable Code

*xhavle03@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

The recent growth of cyber-crime, in particular intrusion into computer networks, has increased the demand for systems for detection of malicious network traffic. In such systems, regular expressions are often used to describe traffic to be selected for a further inspection (e.g., suspicious of containing an attack, tunnelled protocol, ...). The increasing speed of networks requires hardware solutions to accelerate network traffic filtering based on the regular expressions of interest. These hardware solutions implement finite automata corresponding to the regular expressions. The hardware resources are, however, restricted, and so methods for reducing the size of the automata must be used.

The reduction may be based on the classical approach of determinizing and minimizing the automata. This, however, incurs the possibly exponential explosion in the size of the determinized automata. Alternatively, one can use techniques for reducing directly nondeterministic finite automata (NFA) using, e.g., various simulations [1, 2], and further techniques like those proposed in [3] and implemented in the RABIT and Reduce tool[1]. Still, even such reductions need not be sufficient. For example, within our collaboration with the group of accelerated network technologies at FIT BUT (ANT@FIT), which is world-renowned in the area of the hardware accelerated processing of high-speed network traffic, we were given regular expressions that translate to NFAs having from units to tens of thousands of states. Classical determinization and minimization simply explodes on these automata. Techniques of [3] may reduce the automata, though the reduction may not be sufficient.

To improve on the above situation, we propose a novel approach based on an approximate reduction of the automata. Note that the approximate reduction may change the language of the automata, which can, in theory, lead to both false positives and false negatives when classifying the network traffic. This may still be, however, better than not being able to run any classification at all or than having to completely ignore some traffic patterns. Moreover, one can also aim

[1] http://languageinclusion.org/doku.php?id=tools

at the approximate reductions that will solely over-approximate the language. This can then increase the amount of packets from a hardware filtering device to the subsequent final software classification, but no critical traffic needs to be lost this way. In addition, we hope that the reduction techniques that we proposed can be fine-tuned such that a significant reduction of the automata can be achieved without a significant increase in the traffic sent for the final classification in software. Our first experimental results confirm this hypothesis.

Since we deal in this paper with the approximate reductions, suitable methods for comparing similarity of languages are necessary. This is needed so that we can control the reduction in a systematic way. For this reason, we propose using a distance that is expressed as a probability that a randomly chosen string belongs to the symmetric difference of the input languages. A random string is chosen with respect to a probabilistic distribution, which is represented by a probabilistic automaton (PA). This PA is obtained by an analysis of a representative sample of packets that occur in the network flow. Hence, a learned PA gives us a compact and abstract model of the network traffic, i.e., a representation of the frequency of occurrence of various packets in the network.

Subsequently, we propose several automata reductions that are specifically tailored for the use in network traffic monitoring. The reductions are based on either under-approximating the languages of automata by pruning their states, or over-approximating the language by introducing new self-loops (and pruning redundant states later). The reductions can be parametrized by a maximal error, which is given with respect to the desired distance between the language of an input automaton and the language of the reduced automaton. In this paper, we give a high-level description of the used distance and the proposed reductions.

The proposed techniques were implemented and evaluated on a dataset provided by the ANT@FIT group. We have obtained quite promising results showing the potential of the proposed approach, which can be further improved in many ways.

2. Preliminaries

In this section, we give some basic definitions and we briefly describe existing methods for automata reduction and comparison of the similarity of their languages.

Definition 1. *A* (nondeterministic) finite automaton *(NFA) over an alphabet* Σ *is a tuple* $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ *where*

- Q *is a finite non-empty set of states,*
- $\delta : Q \times \Sigma \to 2^Q$ *is a transition function,*
- $q_0 \in Q$ *is an initial state, and*
- $F \subseteq Q$ *is a set of accepting (final) states.*

A finite automaton $A = (Q, \Sigma, \delta, q_0, F)$ *is called* deterministic *(DFA) if* $\forall q \in Q$ *and* $\forall a \in \Sigma : |\delta(q,a)| \leq 1$.

Definition 2. *An NFA is called* unambiguous *(UFA) if it has at most one accepting computation on every input string.*

Finally, we give an informal definition of a probabilistic automaton. A *probabilistic automaton* (PA) is an NFA where transitions are associated with probabilities and final states are associated with a probability of accepting in this state. For each state, the sum of probabilities of all outgoing transitions and the probability of accepting in this state is equal to one.

2.1 Automata Reductions

If we consider DFAs, there exists an efficient algorithm for their minimization. Unfortunately, a use of this algorithm for general NFAs requires a prior determinization of the input automaton, which may cause an exponential increase in the number of states (compared to the input NFA). Moreover, the size after minimization can still be exponentially larger than the size of the input NFA.

An alternative approach is to reduce NFAs directly without determinization. The general NFA state minimization is a PSPACE-hard problem [4], but there still exist some practically feasible algorithms to reduce NFAs. One of those algorithms is based on merging states according to the (maximal) *simulation equivalence relation* on the states. The definition of the simulation equivalence is based on the notion of the forward or backward simulation relation [5].

It can be shown that for each NFA there exists a unique largest simulation, called the *simulation preorder*. The *simulation equivalence* for the simulation preorder $\preccurlyeq$ is then given as $\preccurlyeq \cap \preccurlyeq^{-1}$. Moreover, there exists a polynomial-time algorithm for computing the largest simulation equivalence [6].

For the given simulation equivalence, the state merging algorithm merges all states in the same equivalence class. A generalization of the above mentioned reduction is a reduction based on preorders [1].

The basic simulation reduction uses the only simulation equivalence (forward or backward) for a reduction. Another approach is to use a composition of the forward and the backward equivalences [2]. An automaton is then reduced according to the combined equivalence obtained from the backward and the forward simulation preorders.

The above mentioned methods use only merging of states for reduction. This is not, however, the only possible approach. The state merging methods can be combined with, e.g., a removing of transitions [3] (tools RABIT and Reduce).

So far, we dealt with language-preserving reductions only, where for an NFA $\mathcal{A}$ and a reduced NFA $\mathcal{A}'$ it holds that $L(\mathcal{A}) = L(\mathcal{A}')$. There also exists a way of reducing automata, called *hyperminimization*, which modifies the language of the input automaton. A hyperminimizing algorithm converts an input automaton $\mathcal{A}$ into a smaller automaton $\mathcal{A}'$ such that the symmetric difference between languages $L(\mathcal{A})$ and $L(\mathcal{A}')$ is a finite set [7]. Hyperminimization, however, is not suitable for our purposes because reduction up to a finite difference need not yield a sufficiently small automaton. Moreover, by this reduction, we might remove important strings from the input language.

2.2 Automata Language Comparison

Now, we discuss methods for measuring the difference of languages of finite automata. One of the ways is based on comparing the similarity of strings from the languages as follows. The similarity of two strings can be expressed as a cost of operations transforming a string of a source language to some string of the target language (a symbol insertion, deletion, or substitution by a different symbol). Then the minimal cost of a sequence of these operations transforming a string x into a string y is called the *edit-distance* of strings x and y (denoted as $d(x,y)$) [8]. The notion of edit-distance between strings can be generalized to languages by

$$d(L_1,L_2) = \inf\{d(x,y) \mid x \in L_1, y \in L_2\}. \quad (1)$$

The reason why this definition is not suitable for many applications, including ours, is the fact that if languages L_1 and L_2 have at least one common string, then their edit-distance is zero.

Another approach to comparing regular languages is using the Jaccard distance and the Cesaro-Jaccard distance [9]. The actual computation is, however, quite complicated because of a need to determine the matrix polynomials of the automaton adjacency matrix when computing the Cesaro-Jaccard distance. Moreover, for our purposes, it seems more practical to somehow reflect in the distance the fact that not all strings are of the same importance for us, e.g., some of them appear rarely or do not appear at all. Hence, an approximate reduction that makes a mistake by classifying such strings should be better than on that makes a mistake on more important strings.

3. Probabilistic Language Distance

For a comparison of the languages of automata used in network traffic monitoring we propose the following distance (we call it the *probabilistic language distance*). This distance expresses the probability that languages L_1 and L_2 over the same alphabet Σ differ on a word chosen randomly from Σ^* according to a given distribution on words.

Definition 3. *Let μ be a distribution over Σ^* and its pointwise extension to a set of strings L is defined as $\mu(L) = \sum_{w \in L} \mu(w)$. Further, let L_1 and L_2 be languages over Σ. Then the* probabilistic language distance *$d_\mu : 2^{\Sigma^*} \times 2^{\Sigma^*} \to \mathbb{R}_{\geq 0}$ is defined as*

$$d_\mu(L_1,L_2) = \mu(L_1 \triangle L_2), \quad (2)$$

where $A \triangle B$ denotes the symmetric difference of the sets A and B, defined as $A \triangle B = (A \setminus B) \cup (B \setminus A)$.

We propose an algorithm for a computing probabilistic distance for regular languages. Moreover, if the languages are given as UFAs, our proposed algorithm can compute the distance in a polynomial time. In this case, no prior determinization is necessary. If the languages are given by general NFAs, one can use two possible approaches. The first uses on-the-fly determinization and product construction, and the second uses disambiguation [10]. Both mentioned approaches can be more efficient compared to an *a priori* NFA determinization (but in the worst case, both approaches can run in exponential time).

The probabilistic distance can be used in our network monitoring application for comparing automata representing attacks or communication protocols. In this setting we can use a PA to model a network traffic (more concretely occurrences of packets in the network traffic). Then the probabilistic distance expresses the difference of two NFAs with regard to the traffic model. This is crucial if we are performing operations with NFAs (e.g., their reduction), and we want to give a formal guarantees on the error of the modified automaton.

4. Approximate Reduction of Automata

In this section, we introduce two approaches for the automata reduction. The first one is based on removing branches of an input automaton (we call it the *pruning reduction*). The second one is then based on adding self-loops into the automaton (we call it the *self-loop reduction*). As in the case of the probabilistic distance described in the previous section, for reductions we also use a probabilistic automaton for representing the

input traffic. Based on an input probabilistic automaton, the pruning reduction selects branches to be removed, and the self-loop reduction selects states where a self-loop over every symbol is to be added. The reduction methods are proposed directly for reducing NFAs, however, if an input automaton is unambiguous, more efficient methods for computation can be used. The workflow of automata reductions and obtaining probabilistic automaton in network traffic monitoring is shown in Figure 1.

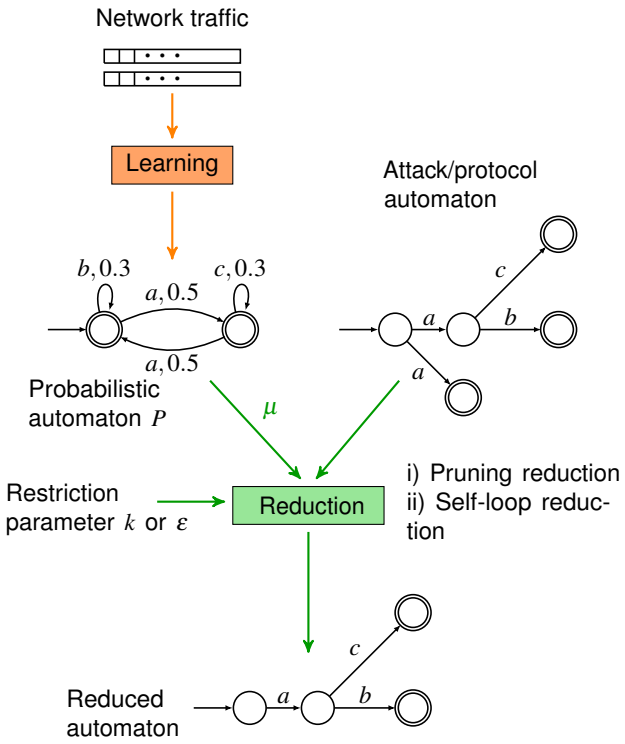

Figure 1. The workflow of the network automata reductions. In the first step, P is synthesized from the input traffic. In the second step, P and an NFA describing attacks or protocols are the input for the reduction. The reduction yields a reduced automaton satisfying the restriction conditions.

4.1 Pruning Reduction

We start with the pruning reduction. As we have already mentioned, the pruning reduction selects branches of the input NFA that are later removed. The choice of branches depends on the input probabilistic automaton. Because this reduction approach is language non-preserving, it is necessary to restrict the reduction by a parameter. Because it removes states, it performs language under-approximation. According to the meaning of this parameter, we divide the pruning reduction to the *ε-pruning reduction*, and the *k-pruning reduction*. In the case of the ε-pruning reduction, the parameter sets a maximum error of the reduction. The error denotes the maximum probabilistic distance of the input NFA and the reduced NFA. In the case of the k-pruning reduction, the parameter restricts the ratio between the number of states of the reduced NFA and the number of states of the original NFA.

The proposed algorithm, in the first step, labels each final state q_f with a probability of the occurrence of some string from the backward language of q_f, i.e. $\mu(L^{-1}(q_f))$ (probability of the backward language). An input automaton is reduced according to a subset of final states F' as follows. The reduced automaton is obtained from the input automaton by removing branches leading to F'. Hence, the algorithm selects (according to the computed labels) a subset of the final states such that the reduced automaton meets the restriction criterion. Finally, the reduced automaton is obtained from the best found subset. However, for the optimal selection of the removed final states, we have to go through all subsets of final states, which can be infeasible for bigger automata. Therefore, we can relax the optimality and perform the pruning reduction on the level of subautomata.

Subautomata Pruning A common automata used in network traffic monitoring consist of several subautomata. We can thus divide the whole automaton into several independent subautomata $\mathcal{A}_1,\ldots,\mathcal{A}_n$, such that their sets of final states $F_1,\ldots,F_n$ form a decomposition of F. Hence, we can perform the relaxed pruning reduction by removing whole subautomata, i.e. we do not select final states for the reduction but we select the whole subautomata. For the optimal selection of subautomata, 0-1 integer linear programming, which is a special form of the integer linear programming where each variable can only take on the value of 0 or 1, can be used.

4.2 Self-loop Reduction

In this section, we present our second approach for reducing finite automata. As in the case of the pruning reduction, the self-loop reduction is a language non-preserving reduction. The self-loop reduction consists of adding self-loops to certain states (and making these states final), followed by removing all other transitions from these states and trimming the modified automaton. Hence, it performs a language over-approximation. The choice of states for adding self-loops depends again on the input probabilistic automaton. The reduction is restricted by the parameter (k or ε) with the same meaning as in the case of the pruning reduction.

In the first step, the proposed algorithm computes for each state of the input automaton the weight of its backward language (the weight of a word w is a probability of w ignoring the probability of accepting). Then,

the algorithm selects states where the self-loop is to be added (according to the computed labels), such that the restriction criterion is met. Because the choice of the optimal set would lead to an exponential-time algorithm, we use a greedy approach, which may not find the optimal solution. On the other side, a choice can be made in a polynomial time.

5. Experiments

The proposed techniques for automata distance computation and their reduction were implemented in a prototype tool. This section provides results of experiments performed with the implemented tool.

For these experiments we use the probabilistic automaton $\mathcal{P}_{2k}$ learned from 2000 packets (algorithm Alergia [11, Chapter 16]). For learning of the probabilistic automaton we use the TREBA tool[2]. We performed the reduction of some selected automata obtained from regular expressions. Then, we computed the probabilistic distance of the input and the reduced automata. We also determined the error related to our captured traffic (a multiset of packets). This error is then computed as a quotient of the number of misclassified packets and the number of all packets.

When the automata are synthesized into hardware, the input string is accepted if a final state is reached during the computational steps (the input string may not be completely processed). Therefore, if we compute the error, we also use this "prefix acceptance" (if a string w is accepted in this way by some NFA $\mathcal{A}$, we denote it as $w \in_{pr} L(\mathcal{A})$). The packet p is thus misclassified iff $p \in_{pr} L(\mathcal{A}) \oplus p \in_{pr} L(\mathcal{A}_r)$ where $\mathcal{A}$ is an original NFA, $\mathcal{A}_r$ is the reduced NFA, and $\oplus$ is logical nonequivalence.

Let us now present more detailed information about reductions of each input automaton (the automata are transformed from REs, which are obtained from the ANT@FIT group)

http-bots The first input automaton is `http-bots`. This automaton has only 8 states. We performed the k-pruning, and the k-self-loop reduction for various values of k. The parameter k denotes the maximum ratio of the number of states of the reduced NFA and the number of states of the original NFA. The results of the reduction are shown in Table 1. The error related to the captured traffic (the column "Traffic error") is obtained from the multiset of 10^6 packets. The error evaluation took about 10 min for each pair of the reduced and the original automaton.

[2] https://code.google.com/archive/p/treba

From the table, we can see that in the case of the self-loop reduction, the computed probabilistic distance is greater than the traffic error, and the difference between the probabilistic distance and the traffic error is quite small. Moreover, observe that we could remove from the original NFA more than half of the states with the traffic error being is less than 1%.

In the case of the k-pruning reduction, the difference between the traffic error and the computed probabilistic distance is bigger. The difference may be caused mainly by an inaccuracy of the learned PA.

This experiment shows that the self-loop reduction is probably more suitable for our application. The automaton obtained via the self-loop reduction decides on the acceptance of an input string after reading some prefix. In our network monitoring application, it can be a quite common phenomenon (we are able to classify some packets after certain prefix of its payload). Therefore, in the further experiments we focus mainly on the self-loop reduction.

Table 1. The reduction of the automaton `http-bots` with 8 states.

(a) The self-loop reduction

k	States	Traffic error	Probabilistic distance
0.0	1	0.992	0.999
0.2	3	0.001 69	0.007 86
0.5	4	0.000 89	0.003 72
0.6	5	6.0×10^{-6}	$6.869\,77 \times 10^{-5}$
0.7	6	0.0	$1.349\,92 \times 10^{-5}$
1.0	8	0.0	0.0

(b) The pruning reduction

k	States	Traffic error	Probabilistic distance
0.0	1	0.007 18	$2.003\,72 \times 10^{-8}$
0.5	4	8.41×10^{-4}	$2.733\,57 \times 10^{-13}$
0.6	5	0.006 34	$2.003\,69 \times 10^{-8}$
0.7	6	0.0	0.0
1.0	8	0.0	0.0

info.rules The second automaton for the reduction is `info.rules` with 16 states. We performed the k-self-loop reduction for various values of k. The results of the reduction are shown in Table 2. The error related to the captured traffic is obtained from the multiset of 10^6 packets, and the evaluation took about 9 hours for each pair of the reduced and the original automaton.

The computed probabilistic distance is again greater than the traffic error. From the table, we can see that for the k-self-loop reduction with the parameter $k = 0.2$ we obtained an automaton, having only 25% of states

of the original automaton, with the traffic error less than 1%.

Table 2. The self-loop reduction of the automaton `info.rules` with 16 states.

k	States	Traffic error	Probabilistic distance
0.0	1	1.0	1.0
0.2	4	0.00861	0.03041
0.5	9	0.0	4.04245×10^{-10}
0.7	12	0.0	1.93657×10^{-12}
1.0	16	0.0	0.0

shellcode.rules The last considered automaton is `shellcode.rules` with 95 states. We again performed the k-self-loop reduction for various values of k. The results are shown in Table 3. The error related to the captured traffic is obtained from the multiset of 5×10^5 packets, and the evaluation took about 12 hours for each pair of the reduced and the original automaton.

For each reduced automaton we give only an upper-bound of the probabilistic distance. This is because the reduced automaton is no longer an unambiguous automaton. For the computation of the exact distance, we therefore need a prior disambiguation of the reduced automaton, which would yield a huge automaton.

The computed probabilistic distance is not greater than the traffic error for automata reduced with the parameters $k = 0.3$ and $k = 0.5$. This is caused by the inaccuracy of the learned PA.

Table 3. The self-loop reduction of the automaton `shellcode.rules` with 95 states.

k	States	Traffic error	Probabilistic distance
0.0	1	1.0	≤ 1.0
0.3	29	1.6×10^{-5}	$\leq 1.27877 \times 10^{-12}$
0.5	48	1.4×10^{-5}	$\leq 6.41914 \times 10^{-19}$
0.7	67	0.0	$\leq 5.29250 \times 10^{-25}$
1.0	95	0.0	0.0

6. Conclusion

In this paper, we gave a high-level description of proposed methods for comparing languages and automata reductions used in network traffic monitoring. The proposed methods are based on the knowledge of an input traffic represented by a probabilistic automaton. According to a PA, we select automata branches for removing, and states where self-loops are added.

The proposed methods are implemented and evaluated on a set of small automata used in network traffic monitoring. The experiments on these automata show that we are able to reduce input automata to less than 25% of their size with the traffic error less than 1%. Currently, the most time-demanding operation of these experiments is the traffic error evaluation. This problem could be solved by a hardware accelerated evaluation.

The performed experiments also indicate that a success of the reduction depends a lot on the used probabilistic automaton. Therefore, in the further continuation of the work we closely focus on the learning process. The very first ideas, for instance, preprocessing of the captured traffic before learning, may lead to a more precise probabilistic automata.

Acknowledgement

I would like to thank my supervisor Tomáš Vojnar, and my consultants Ondra Lengál and Milan Češka jr. for their time, ideas, and numerous advice.

References

[1] Lucian Ilie, Gonzalo Navarro, and Sheng Yu. *On NFA Reductions*, pages 112–124. Springer, Berlin, Heidelberg, 2004.

[2] Parosh A. Abdulla, Lukáš Holík, Lisa Kaati, and Tomáš Vojnar. A Uniform (Bi-)Simulation-Based Framework for Reducing Tree Automata. *Electronic Notes in Theoretical Computer Science*, 251:27–48, 2009.

[3] Richard Mayr and Lorenzo Clemente. Advanced Automata Minimization. In *Proceedings of the 40th Annual ACM Symposium on Principles of Programming Languages*, POPL '13, pages 63–74, New York, NY, USA, 2013. ACM.

[4] Tao Jiang and B. Ravikumar. Minimal NFA Problems are Hard. *SIAM Journal on Computing*, 22(6):1117–1141, 1993.

[5] Lukáš Holík. *Simulations and Antichains for Efficient Handling of Finite Automata*. PhD thesis, FIT VUT v Brně, Brno, 2010.

[6] Lucian Ilie and Sheng Yu. Algorithms for Computing Small NFAs. In *Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science*, MFCS '02, pages 328–340, London, UK, 2002. Springer.

[7] Andrew Badr, Viliam Geffert, and Ian Shipman. Hyper-minimizing minimized deterministic finite state automata. *RAIRO - Theoretical Informatics and Applications*, 43(1):69–94, 2009.

[8] Mehryar Mohri. Edit-Distance of Weighted Automata: General Definitions and Algorithms. *International Journal of Foundations of Computer Science*, 14(06):957–982, 2003.

[9] Austin J. Parker, Kelly B. Yancey, and Matthew P. Yancey. Regular Language Distance and Entropy. *CoRR*, abs/1602.07715, 2016.

[10] Mehryar Mohri. *A Disambiguation Algorithm for Finite Automata and Functional Transducers*, pages 265–277. Springer, Berlin, Heidelberg, 2012.

[11] Colin de la Higuera. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, New York, NY, USA, 2010.