BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

http://fit.vut.cz

Project practice
2022/2023

Utilization of Repeating Substructures for Efficient Representation of Automata

Bc. Michal Šedý*

Abstract
Nondeterministic finite automata (NFA) are widely used across almost all fields of computer science, such as for a representation of regular expressions, monitoring high-speed networks, in abstract regular model checking, program verification, or in decision procedures of WS1S and WS2S logics. NFAs are even used in bioinformatics for searching sequences of nucleotides in DNA. The basic technique for reducing computational resources (memory, time, or the amount of hardware components) when working with NFAs is minimization. The most well-known minimization methods are state merging and transition pruning. Although combining these two methods can reduce the size of an automaton by up to 50%, the resulting automaton can still contain duplicate (similar) transition sequences. There are even automata that cannot be minimized by these methods. This work presents a new way of automata minimization based on a transformation of a NFA into a nondeterministic pushdown automaton (NPDA). The transformation identifies and represents the most similar parts (procedures) of the automaton only once. This algorithm enables the reduction of previously non-minimalizable automata and also improves the results of other minimizations. The principle of transforming NFA into NPDA can be understood as a transformation of a purely sequential program into a program with mutually communicating procedures.

Keywords: Nondeterministic Finite Automata — Minimization — Nondeterministic Pushdown Automata — Regular Expressions — Language Similarity

*xsedym02@stud.fit.vut.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Nondeterministic finite automata (NFA) were introduced by Michael Rabin and Dana Scott in [13]. Unlike their deterministic counterpart, NFAs have the ability to make a transition to multiple states based on the same input symbol. This allows NFAs to represent the language more compactly. However, this feature makes the minimization of NFAs difficult. Despite the difficulty of minimization, NFAs are widely used in almost all fields of computer science, such as representing regular expressions, monitoring high-speed networks [14], in abstract regular model checking [5], in verifying programs that manipulate strings [2] or in decision procedures in the WS1S and WS2S logic [10, 9]. NFAs are even used in bioinformatics to search for nucleotide sequences in DNA [3].

The basic technique for reducing computational requirements when working with NFA is minimization. The most well-known minimization technique is a state merging [4, 6, 12], which searches for two language equivalent states and then merges them into one. Another approach is transition pruning [6, 8], which removes transitions from the state with weaker language based on language inclusion. The opposite of transition pruning is transition adding (saturation) [4, 8], where newly added transitions may allow further state merging or transition pruning.

The mentioned minimization methods reduce the size of most automata by up to 50%, but duplicitous transition sequences still exist in the resulting automata. There are also types of automata that cannot be minimized by current methods, such as automata with a linear structure (without branching) or automata representing words with a given infix, same prefix, and suffix (e.g. *abba, cbbc*). In these cases, state merging, transition pruning, and saturation cannot minimize the automata, because these reduction methods are based on language inclusions and these types of automata do not contain any. In the structure of these automata, only similar transition segments occur (e.g. *bb* from the previous example).

This work describes a new minimization algorithm that uses the similarity of transition sequences in order to reduce the size of the automaton. The approach involves the informed conversion of a nondeterministic finite automaton (NFA) to a nondeterministic pushdown automaton (NPDA). In the NPDA, similar sequences can be replaced by one procedure that uses the symbol stored on the stack to determine the branch of the original automaton where the calculation is located. The goal of a successful transformation is to replace such sequences of transitions that the savings from their reduction will exceed the overhead of the stack operations. The reduction of automata representing regular expressions can be improved on average by 32.1% using this minimization approach.

The transformation of NFA to NPDA can be understood as the conversion of a purely sequential program to a program using communicating procedures. Like the NPDA, a call stack is used in the program to maintain information about the current branch and where the calculation should return to after the procedure ends.

2. Preliminaries

This section defines the fundamental terms utilized in our reduction approach, such as nondeterministic finite automaton, nondeterministic push down automaton. Additionally, it introduces the simulation relation and the product of a NPDA, which represents all possible procedure candidates.

2.1 Nondeterministic Finite Automaton

A *Nondeterministic Finite Automaton* (NFA) is a 5-tuple $M = (Q, \Sigma, \delta, I, F)$, where:

- Q is a finite set of states,
- Σ is an alphabet,
- $\delta : Q \times \Sigma \rightarrow 2^Q$ is a transition function,
- $I \subseteq Q$ is a set of initial states, and
- $F \subseteq Q$ is a set of final states.

The transition function δ can be generalized for a set of symbols. Let $q \in Q$ and $A \subseteq \Sigma$, then $\delta(q, A) = \bigcup_{a \in A} \delta(q, a)$.

We also define the inverse function δ^{-1}, where: $q \in \delta^{-1}(r, a) \iff r \in \delta(q, a)$, for $a \in \Sigma$ and $q, r \in Q$.

Configuration

A *configuration* of a NFA is an ordered pair $C \in Q \times \Sigma^*$. $(q, w) \in C$ indicates that the automaton is in state q and that the unprocessed string w remains on the input.

Transition

A *transition* is a binary relation $\vdash \subseteq C \times C$, defined as follows: $(q, w) \vdash (r, w') \iff w = aw' \wedge r \in \delta(q, a)$, for $q, r \in Q$, $w, w' \in \Sigma^*$, $a \in \Sigma$.

Language

The *forward language* of state $q \in Q$ is the set $\overrightarrow{L}(q) = \{w \in \Sigma^* \mid (q, w) \vdash (f, \varepsilon)$, where $f \in F\}$.

The *backward language* of state $q \in Q$ is the set $\overleftarrow{L}(q) = \{w \in \Sigma^* \mid (i, w) \vdash (q, \varepsilon)$, where $i \in I\}$.

The *language of the automaton M* is the set defined as $L(M) = \bigcup_{i \in I} \overrightarrow{L}(i)$.

2.2 Nondeterministic Pushdown Automaton

A *Nondeterministic Pushdown Automaton* (NPDA) is an 8-tuple $M = (Q, \Sigma_\varepsilon, \Gamma_\varepsilon, \delta, I, \lambda, F, \phi)$, where:

- Q is a finite set of states,
- Σ_ε is an alphabet containing symbol ε,
- Γ_ε is a stack alphabet containing symbol ε,
- $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow 2^{Q \times \Gamma_\varepsilon}$ is a transition function,
- $I \subseteq Q$ is a set of initial states,
- $\lambda : I \rightarrow 2^{\Gamma_\varepsilon} \setminus \{\emptyset\}$ is a function of initial stack symbols,
- $F \subseteq Q$ is a set of final states, and
- $\phi : F \rightarrow 2^{\Gamma_\varepsilon} \setminus \{\emptyset\}$ is a function of final stack symbols.

The function λ specifies the initial stack symbol for an initial state. Let $i \in I$. If $\lambda(i) = \{\varepsilon\}$, then the automaton starts in state i with an empty stack. On the other hand, if $\lambda(i) = G \neq \{\varepsilon\}$, then the automaton can start in state i with a nondeterministically chosen symbol from set G.

The function ϕ specifies the acceptance combination of a state and a stack symbol. Let $f \in F$. If $\phi(f) = \{\varepsilon\}$, then the automaton accepts in state f with an empty stack. Conversely, if $\phi(f) = G \neq \{\varepsilon\}$, then the automaton accepts in state f if the symbol g on the top of the stack belongs to set G.

Similarly to NFA, a transition function can also be defined for a set of symbols $A \subseteq \Sigma_\varepsilon$ or $B \subseteq \Gamma_\varepsilon$.

The inverse transition function δ^{-1} of NPDA, is defined as: $(q, \beta) \in \delta^{-1}(r, a, \gamma) \iff (r, \gamma) \in \delta(q, a, \beta)$, where $q, r \in Q$, $a \in \Sigma_\varepsilon$, $\beta, \gamma \in \Gamma_\varepsilon$. Note that in the inverse transition function, the push and pop stack symbols are swapped.

Configuration

A *configuration* of a NFA is an ordered triple $C \in Q \times \Sigma^* \times \Gamma^*$, where $(q,w,\beta) \in C$ means that the machine is in the state q, the remaining unprocessed string is w, and the stack contains the string β (the top of the stack is on the left).

Transition

A *transition* is a binary relation $\vdash \subseteq C \times C$, where: $(q,w,\beta) \vdash (r,w',\beta') \iff w = aw' \wedge \beta = X\alpha \wedge \beta' = Y\alpha \wedge (r,Y) \in \delta(q,a,X)$, for $q,r \in Q$, $w,w' \in \Sigma^*$, $a \in \Sigma_\varepsilon$, $X,Y \in \Gamma_\varepsilon$.

Language

A *forward language* of state $q \in Q$ with a stack $\alpha \in \Gamma^*$ is a set $\overrightarrow{L}(q,\alpha) = \{w \in \Sigma^* \,|\, (q,w,\alpha) \vdash (f,\varepsilon,\beta) \wedge \beta \in \phi(f)$, where $\beta \in \Gamma_\varepsilon, f \in F\}$.

A *language of the automaton* M is the set defined as $L(M) = \bigcup_{i \in I} \bigcup_{\alpha \in \lambda(i)} \overrightarrow{L}(i,\alpha)$.

Transition Alphabet

Transition Alphabet of a transition between states q and r from Q is a set $\sigma(q,r) = \{a \in \Sigma \,|\, \exists \alpha,\beta \in \Gamma_\varepsilon : (r,\beta) \in \delta(q,a,\alpha)\}$. It is the set of symbols from Σ that label the transitions from state q to state r.

Epsilon Transition Alphabet of a transition between states q and r from Q is a set $\sigma_\varepsilon(q,r) = \{a \in \Sigma \,|\, (r,\varepsilon) \in \delta(q,a,\varepsilon)\}$. For each symbol in this set, there exists a transition that goes from state q to r without access the stack.

Nonepsion Transition Alphabet of a transition between states q and r from Q is a set $\overline{\sigma_\varepsilon}(q,r) = \sigma(q,r) \setminus \sigma_\varepsilon(q,r)$. It is a complement of the Epsilon Transition Alphabet.

Stack Alphabet of State

Stack Alphabet of State $q \in Q$ is a set $\gamma(q) = \{\alpha \in \Gamma \setminus \{\varepsilon\} \,|\, (i,w,\beta) \vdash^* (r,w',\alpha\beta')$, where $i \in I, r \in Q, w,w' \in \Sigma^*, \beta \in \lambda(i), \beta' \in \Gamma^*\}$. It is a set of stack symbols that can appear on the top of the stack while the automaton is in the state q.

2.3 Representation of NFA using NPDA

The conversion of a NFA to its language equivalent NPDA is the initial step in our reduction approach that employs procedure mapping, and it is carried out as follows.

Fore every NFA $M_F = (Q_F,\Sigma_F,\delta_F,I_F,F_F)$ there exists its language equivalent $((M_F) \equiv L(M_P))$ NPDA $M_P = (Q_P,\Sigma_P,\Gamma_\varepsilon,\delta_P,I_P,\lambda,F_P,\phi)$, where:

- $Q_P = Q_F$,
- $\Sigma_P = \Sigma_F$,
- $\Gamma_\varepsilon = \{\varepsilon\}$,
- $\delta_P(q,a,\varepsilon) = \delta_F(q,a) \times \{\varepsilon\}$ for $q \in Q_P, a \in \Sigma_P$,
- $I_P = I_F$,
- $\lambda(i) = \{\varepsilon\}$ for $i \in I$,
- $F_P = F_F$, and
- $\phi(f) = \{\varepsilon\}$ for $f \in F$.

In this work, only nondeterministic pushdown automata with a stack limited to maximum one symbol ($NPDA_1$) is used. For each $NPDA_1$ holds, that $L(NPDA_1) \subseteq \mathscr{L}_3$.

Proof. Let $M = (Q,\Sigma,\Gamma_\varepsilon,\delta,I,\lambda,F,\phi)$ be the NPDA$_1$. For any finite input $w \in \Sigma^*$, the set of all possible configurations is defined by cartesian product $Q \times \{u' \in \Sigma^* \,|\, u'v' = w$, where $v' \in \Sigma^*\} \times \Gamma_\varepsilon$. As the sets ($Q$, the set of suffixes of w, and Γ_ε) that create the configurations are all finite, the number of configurations is also finite. Hence, NPDA$_1$ can be represented as a NFA, which implies $L(\text{NPDA}_1) \subseteq \mathscr{L}_3$. □

2.4 Subproduct of a NPDA

The subproduct of a NPDA enables the identification of states with a sequence of transitions that exhibit a language intersection (i.e., they are similar). To some extent, the longer the sequence, the greater the reduction potential. These sequences serve as potential candidates for procedure mapping. To evaluate the reduction potential of a procedure, the gain function will be introduced.

Each procedure can consist of up to four types of transitions: call transitions that initiate the procedure, return transitions that conclude the procedure, procedure transitions that embody the procedure itself, and auxiliary transitions that occur within the procedure but are not considered as procedure transitions.

Let $M = (Q,\Sigma,\Gamma_\varepsilon,\delta,I,\lambda,F,\phi)$ be a NPDA. The *Subproduct of a NPDA* is a simple oriented graph $G = (V,E)$, where:

- $V \subseteq \{(r,s) \in Q \times Q \,|\, \sigma(r) \cap \sigma(s) \subseteq \{\varepsilon\}\}$, is a set of vertices and
- $E \subseteq \{(u,v) \in V \times V \,|\, u \neq v\}$, is a set of edges.

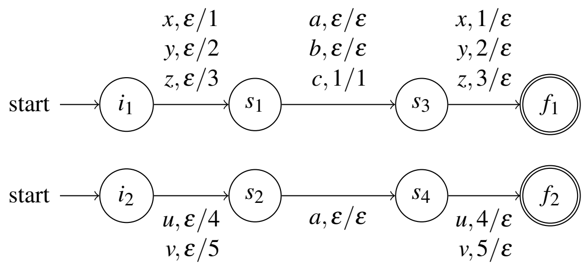
Edge Gain

Each edge in the subproduct of a NPDA has a gain assigned by a function $G_e : E \to \mathbb{Z}$, such that:

$$\begin{aligned} G_e((r,r'),(s,s')) = {} & |\sigma_\varepsilon(r,s) \cap \sigma_\varepsilon(r',s')| - \\ & |\sigma_\varepsilon(r,s) \setminus \sigma_\varepsilon(r',s')| \cdot max(0,|\gamma(r) - 1|) - \\ & |\sigma_\varepsilon(r',s') \setminus \sigma_\varepsilon(r,s)| \cdot max(0,|\gamma(r') - 1|) \end{aligned}$$

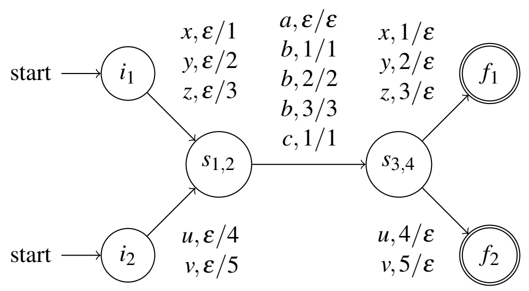
The gain of an edge $((r,r'),(s,s'))$ represents the difference in the number of transitions before and after the creation of the procedure based on the transitions between the states r,s and r',s'.

It is evident that the gain can be negative, due to the increase in the number of transitions that tests stack symbol, as shown in Figure 1.



$x,\varepsilon/1$ $y,\varepsilon/2$ $z,\varepsilon/3$
$a,\varepsilon/\varepsilon$ $b,\varepsilon/\varepsilon$ $c,1/1$
$x,1/\varepsilon$ $y,2/\varepsilon$ $z,3/\varepsilon$
start
i_1
s_1
s_3
f_1
start
i_2
s_2
s_4
f_2
$u,\varepsilon/4$ $v,\varepsilon/5$
$a,\varepsilon/\varepsilon$
$u,4/\varepsilon$ $v,5/\varepsilon$

NPDA before procedure mapping with 14 transitions.



$x,\varepsilon/1$ $y,\varepsilon/2$ $z,\varepsilon/3$
$a,\varepsilon/\varepsilon$ $b,1/1$ $b,2/2$ $b,3/3$ $c,1/1$
$x,1/\varepsilon$ $y,2/\varepsilon$ $z,3/\varepsilon$
start
i_1
f_1
$s_{1,2}$
$s_{3,4}$
start
i_2
f_2
$u,\varepsilon/4$ $v,\varepsilon/5$
$u,4/\varepsilon$ $v,5/\varepsilon$

NPDA after procedure mapping with new transitions testing the stack symbol (15 transitions in total).

Figure 1. This figure illustrates that the creation of a procedure from states s_1, s_2, s_3, and s_4 results in a negative gain of -1 due to the growing number of new transitions.

Vertex Gain

Similar to the edges, each vertex is assigned a gain by a function $G_v : V \to \mathbb{Z}$ such that $G_v(u) = G_e(u,u)$. The vertex gain only reflects the similarity of self-loops in the procedure.

Subproduct creation

Let $M = (Q, \Sigma, \Gamma_\varepsilon, \delta, I, \lambda, F, \phi)$ be a NPDA. The subproduct of M is $G = (V, E)$, where:

- $V = \{(r,s) \in Q \times Q \,|\, \sigma(r) \cap \sigma(s) \subseteq \{\varepsilon\}\}$ and
- $E = \{(u,v) \in V \times V \,|\, u \neq v \wedge G_e(u,v) > 0\}$.

Note that only edges with positive gain are considered in this subproduct of the automaton.

The *degree of a node* $v \in V$ is calculated as the number of edges related to the node, given by a function $d : V \to \mathbb{N}_0$ such that $d(v) = |\{(u,v) \in E \,|\, u \in V\}| + |\{(v,u) \in E \,|\, u \in V\}|$. Since nodes with zero degree are unnecessary in the subproduct of the automaton, the subproduct $G = (V,E)$ can be simplified to $G_s = (\{v \in V \,|\, d(v) > 0\}, E)$.

Procedure Candidate

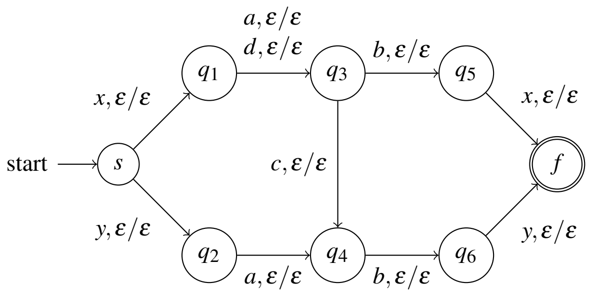
The subproduct of an automaton is an oriented graph $G = (V,E)$, which represents all possible procedure candidates (with a positive gain). Only the most suitable procedure will be selected in each iteration of the reduction algorithm and mapped to the automaton.

Let $G = (V,E)$ be the subproduct of a NPDA. A *procedure candidate* $P = (V_P, E_P)$ is defined as an acyclic linear subgraph of G that satisfies the following conditions:

- $\forall (r,r'),(s,s') : \{r,r'\} \cap \{s,s'\} \neq \emptyset \iff (r,r') = (s,s')$ (each state can be used at most once),
- $\{v_0, v_1, \ldots, v_n\} = V_P$, where $n \in \mathbb{N}$, and
- $\{(v_{i-1}, v_i) \in E \,|\, 1 \leq i \leq n\} = E_P$.

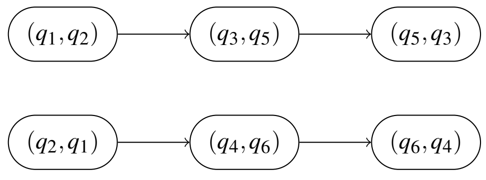
The *states of a procedure candidate* $P = (V,E)$ are defined as the set of all u and v such that: $states(P) = \{u \,|\, (u,v) \in V\} \cup \{v \,|\, (u,v) \in V\}$.

The *gain of a procedure candidate* $P = (V_P, E_P)$ is calculated as the sum of the gains of its nodes and edges: $G_p(P) = \sum_{v \in V_P} G_v(v) + \sum_{(u,v) \in E_P} G_e(u,v)$. The goal is to map the procedure with the highest gain in each iteration.



$a,\varepsilon/\varepsilon$ $d,\varepsilon/\varepsilon$
$b,\varepsilon/\varepsilon$
q_1
q_3
q_5
$x,\varepsilon/\varepsilon$
$x,\varepsilon/\varepsilon$
start
s
$c,\varepsilon/\varepsilon$
f
$y,\varepsilon/\varepsilon$
$y,\varepsilon/\varepsilon$
q_2
q_4
q_6
$a,\varepsilon/\varepsilon$
$b,\varepsilon/\varepsilon$

An example of NPDA.



Supbroduct of the NPDA above.

Figure 2. This figure illustrates a NPDA and its subproduct. The highest yield achieved through the procedure mapping based on this subproduct is 2.

The following subsections provide definitions for all four types of transitions. We consider an NPDA $M = (Q, \Sigma, \Gamma_\varepsilon, \delta, I, \lambda, F, \phi)$ and a procedure $P = (V,E)$ derived from a subproduct of M.

Call Transition

A transition that enters a procedure is referred to as to call transition. The set of *call transitions* for an automaton M and a procedure candidate P is defined as follows: $\delta_{call}(M,P) = \{(q,a,\alpha,\beta,r) \mid (r,\beta) \in \delta(q,a,\alpha)$, where $q \in Q \setminus states(P), r \in states(P), a \in \Sigma, \alpha,\beta \in \Gamma_\varepsilon\}$.

For example, for the automaton M in Figure 2 and the maximum possible procedure candidate P, the set of call transitions is $\delta_{call}(M,P) = \{(s,x,\varepsilon,\varepsilon,q_1),(s,y,\varepsilon,\varepsilon,q_2)\}$.

Return Transition

A transition that exits a procedure is called a return transition. The set of *return transitions* for an automaton M and a procedure candidate P is defined as: $\delta_{ret}(M,P) = \{(q,a,\alpha,\beta,r) \mid (r,\beta) \in \delta(q,a,\alpha)$, where $q \in states(P), r \in Q \setminus states(P), a \in \Sigma, \alpha,\beta \in \Gamma_\varepsilon\}$.

For instance, for the automaton M in Figure 2 and the maximum possible procedure candidate P, the set of return transitions is $\delta_{ret}(M,P) = \{(q_5,x,\varepsilon,\varepsilon,f),(q_6,y,\varepsilon,\varepsilon,f)\}$.

Procedure Transition

Procedure transitions are defined differently from the others. A *procedure transition* uses vertices from a procedure candidate. The set of *procedure transitions* for an automaton M and procedure candidate $P = (V,E)$ is $\delta_{proc}(M,P) = \{((r,r'),a,\alpha,\beta,(s,s')) \mid ((s,\beta) \in \delta(r,a,\alpha) \vee (s',\beta) \in \delta(r',a,\alpha)) \wedge \sigma_\varepsilon(r,s) \cap \sigma_\varepsilon(r',s') \neq \emptyset\}$.

The set of procedure transitions for the automaton M in Figure 2 and the maximum procedure candidate P is $\delta_{proc}(M,P) = \{((q_1,q_2),a,\varepsilon,\varepsilon,(q_3,q_4)),((q_1,q_2),d,\varepsilon,\varepsilon,(q_3,q_4)),((q_3,q_4),b,\varepsilon,\varepsilon,(q_5,q_6))\}$.

Auxiliary Transition

A transition that goes between two procedure states but is not procedure transition is called an auxiliary transition. The set of auxiliary transitions for the automaton M and procedure candidate P is defined as $\delta_{aux}(M,P) = \{(q,a,\alpha,\beta,r) \mid (r,\beta) \in \delta(q,a,\alpha)$, where $q,r \in states(P), a \in \Sigma, \alpha,\beta \in \Gamma_\varepsilon\} \setminus T$, where the $T = \{(r,a,\alpha,\beta,s) \mid ((r,r'),a,\alpha,\beta,(s,s')) \in \delta_{proc}(M,P)\} \cup \{(r',a,\alpha,\beta,s') \mid ((r,r'),a,\alpha,\beta,(s,s')) \in \delta_{proc}(M,P)\}$.

The set of auxiliary transitions for the automaton M shown in Figure 2 and the maximum possible procedure candidate P is $\delta_{aux}(M,P) = \{(q_3,c,\varepsilon,\varepsilon,q_4)\}$.

2.5 Simulation

Simulation[11, 1] on a NFA M is a preorder $\preceq \subseteq Q \times Q$. Let $a \in \Sigma$. The relation $p \preceq q$ only exists if $r \in F \implies q \in F$ and for every $r' \in \delta(r,a)$, there exists $q' \in \delta(q,a)$, which must further satisfy $r' \preceq q'$.

The simulation relation is commonly used as an approximation of the language relation. If state r is simulated by state q ($r \overrightarrow{\preceq} q$), then the language of state r is included in the language of state q ($\overrightarrow{L}(r) \subseteq \overrightarrow{L}(q)$), but not vice versa.

3. Minimization Techniques

This section lists the most commonly used minimization methods, namely: State Merging, Transition Pruning, and Saturation.

3.1 Strate Merging

Two states p and q can be merged into one if at least one of the following conditions is met:

- $\overleftarrow{L}(p) \subseteq \overleftarrow{L}(q) \wedge \overleftarrow{L}(q) \subseteq \overleftarrow{L}(p)$,
- $\overrightarrow{L}(p) \subseteq \overrightarrow{L}(q) \wedge \overrightarrow{L}(q) \subseteq \overrightarrow{L}(p)$, or
- $\overleftarrow{L}(p) \subseteq \overleftarrow{L}(q) \wedge \overrightarrow{L}(p) \subseteq \overrightarrow{L}(q)$.

3.2 Transition Pruning

The transition $pa \rightarrow q$[1] can be removed if one of the following conditions is met:

- $\exists ra \rightarrow q \wedge \overrightarrow{L}(p) \subseteq \overrightarrow{L}(q)$,
- $\exists qa \rightarrow p \wedge \overleftarrow{L}(r) \subseteq \overleftarrow{L}(q)$, or
- $\exists r'a \rightarrow p' \wedge \overleftarrow{L}(r) \subseteq \overleftarrow{L}(r') \wedge \overrightarrow{L}(p) \subseteq \overrightarrow{L}(p')$.

3.3 Saturation

The basic idea of saturation (or transition adding) is an analogy to transition pruning. The transition $pa \rightarrow q$ can be added to the automaton if one of the following conditions is met:

- $\exists qa \rightarrow r \wedge \overleftarrow{L}(p) \subseteq \overleftarrow{L}(q)$, or
- $\exists pa \rightarrow q \wedge \overrightarrow{L}(r) \subseteq \overrightarrow{L}(q)$.

3.4 The Limitation

It can be seen that all minimization techniques are based on language inclusions (can be determined based on simulation). Then in the case of linear automata (without branching), or automata with the same prefix and suffix, these methods cannot be fully utilized, because in these automata, there is a minimum of states in language inclusion.

The illustration in Figure 3 shows that in the automaton, no minimization based on language inclusions can be performed. However, it can be observed that the infix part of the word (unnecessarily) repeats. The main motivation for the work is the fact that in automata of this and similar types, long repeating sequences of transitions occur, which can only be represented once.

[1] Instead of $q \in \delta(p,a)$ we can write $pa \rightarrow q$.

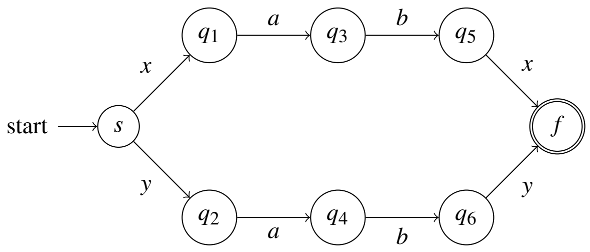


Figure 3. An example of an automaton accepting a word with the infix *ab* where both the prefix and suffix must be *x* or *y*.

4. Conversion of NFA to NPDA

This section describes the algorithm for the optimal conversion of a NFA into NPDA through the creation of procedures for similar transition sequences. In each iteration, the algorithm selects the best procedure candidate (with the highest gain) from the automaton subproduct and maps its edges to a procedure. The conversion process ends when the automaton subproduct is empty and the final automaton is reached.

It is important to clarify what is meant by optimal conversion and how the size of the automaton is measured before discussing the algorithm in detail.

4.1 Conversion Efficiency

It is evident that each NFA can be transformed into a NPDA by utilizing a single state and a stack limited to a maximum of one symbol. However, this method of transformation is not considered to be the optimal solution.

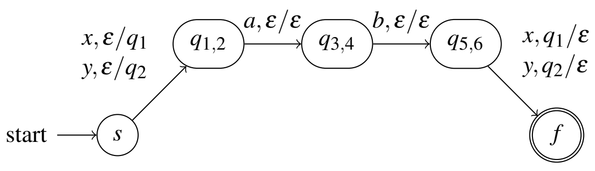
Proof. Let $M = (Q_M, \Sigma_M, \delta_M, I_M, F_M)$ be a NFA. For the automaton M exists NPDA $N = (Q_N, \Sigma_N, \Gamma_\varepsilon, \delta_N, I_N, \lambda, F_N, \phi)$, such that:

- $Q_N = \{q\}$,
- $\Sigma_N = \Sigma_M$,
- $\Gamma_\varepsilon = Q_M \cup \{\varepsilon\}$,
- $\delta_N(q, a, \alpha) = (q, \beta) \iff \beta \in \delta_M(\alpha, a)$,
- $I_N = Q_N = \{q\}$,
- $\lambda(q) = I_M$,
- $F_N = Q_N = \{q\}$,
- $\phi(q) = F_M$.

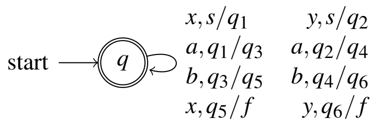
$\square$

The prior proof highlights the ease with which the number of automaton states can be reduced by using the stack to retain information of the current state of computation. Hence, the number of states holds limited significance in evaluating the optimization of the conversion. The crucial measurement for the optimal transformation of a NFA into a NPDA is the number of transitions. If the reduction does not decrease the number of transitions, then it can be said that the conversion has not effectively reduced the size of the automaton.

In order to attain an efficient conversion with maximal reduction in the size of the automaton, the algorithm maps only those transitions that have a positive gain.



The optimally converted automaton from Figure 3.



The NPDA, consisting of a single state, created from the NFA in Figure 3 representing a suboptimal conversion.

Figure 4. The Figure illustrates the contrast between an optimal and suboptimal conversion of the NFA into a NPDA. The optimal NPDA has 5 states, 6 transitions, and utilizes 2 stack symbols, whereas the suboptimal NPDA, despite having only 1 state, has 8 transitions and requires 6 stack symbols.

4.2 Automaton Conversion

The conversion of the NFA M starts with transforming it into the equivalent NPDA N, as has been described in Section 2.3. The subproduct of automaton N represents all potential procedures, where only the best candidate (with the highest gain) is selected for the mapping. The subproduct of the automaton is recalculated each time a procedure is formed. The algorithm terminates when there are no more procedure candidates (the subproduct of the automaton is empty), and the resulting automaton N is returned.

Algorithm 1: Conversion of NFA to NPDA

```
Input: NFA $M = (Q_M, \Sigma_M, \delta_M, I_M, F_M)$
Output: NPDA $N = (Q, \Sigma, \Gamma_\varepsilon, \delta, I, \lambda, F, \phi)$, such that
        $L(N) \equiv L(M)$
1 $N \leftarrow equivalentNPDA(M)$          // see 2.3
2 $Prod \leftarrow subproduct(N)$
3 while $Prod \neq \emptyset$ do
      /* Find a procedure with the
         maximal gain.                   */
4     $P \leftarrow argmax_{P \subseteq Prod}(G_p(P))$
5     $N \leftarrow createProcedure(N, Prod, P)$
6     $N.removeStates(states(P))$
7     $Prod \leftarrow subproduct(N)$
8 return $N$
```

4.3 Procedure Creation

Let the algorithm 1 selects the procedure candidate $P = (V_P, E_P)$ for mapping in automaton M. The following algorithm creates a new state for each pair $(r, r') \in V_P$ and assigns a stack symbol for the branches that pass through states r and state r' if the symbol hasn't been assigned yet in previous runs of the mapping algorithm. The function $procS : Q^M \to Q$ maps the original states r and r' to their respective newly created procedure states. The function $stack : Q \to 2^{\Gamma_\varepsilon}$ maps each state from $states(P)$ to all possible stack symbols that can be used within a procedure state in the future. After this initiation, the procedure is constructed from all related transitions, initial states, and final states. The algorithm returns the newly created NPDA automaton N, where $L(M) \equiv L(N)$.

Algorithm 2: createProcedure

```
Input: NPDA $M = (Q^M, \Sigma, \Gamma_\varepsilon^M, \delta^M, I^M, \lambda^M, F^M, \phi^M)$,
       subproduct $Prod = (V, E)$ of $M$, and a procedure
       $P = (V_P, E_P) \subseteq Prod$
Output: NPDA $N = (Q, \Sigma, \Gamma_\varepsilon, \delta, I, \lambda, F, \phi)$, such that
        $L(N) \equiv L(M)$

1  $N \leftarrow M$
   // Determine stack symbols
2  $(u, v) \leftarrow root(P)$
3  $stackU \leftarrow \{u\}$
4  $stackV \leftarrow \{v\}$
5  if $\gamma(u) \neq \{\varepsilon\}$ then
6      $stackU \leftarrow \gamma(u)$
7  else
8      $\Gamma_\varepsilon \leftarrow \Gamma_\varepsilon \cup \{u\}$
9  if $\gamma(v) \neq \{\varepsilon\}$ then
10     $stackV \leftarrow \gamma(v)$
11 else
12     $\Gamma_\varepsilon \leftarrow \Gamma_\varepsilon \cup \{v\}$

13 $procS : Q \to Q$
14 $stack : Q \to 2^{\Gamma_\varepsilon}$
   // Create procedure states
15 forall $(u, v) \in V_P$ do
16     $uv \leftarrow N.createNewState()$
17     $Q \leftarrow Q \cup \{uv\}$
18     $procS(u) \leftarrow uv$
19     $procS(v) \leftarrow uv$
20     $stack(u) \leftarrow stackU$
21     $stack(v) \leftarrow stackV$
22     $stack(uv) \leftarrow stack(u) \cup stack(v)$
   // Map procedure
23 $N \leftarrow mapCallT(N, \delta_{call}(Prod, P), stack, procS)$
24 $N \leftarrow mapRetT(N, \delta_{ret}(Prod, P), stack, procS)$
25 $N \leftarrow mapProcT(N, \delta_{proc}(Prod, P), stack, procS)$
26 $N \leftarrow mapAuxT(N, \delta_{aux}(Prod, P), stack, procS)$
27 $N \leftarrow mapInitS(N, P, stackU, stackV)$
28 $N \leftarrow mapFinS(N, P, stackU, stackV)$
29 return $N$
```

Call Transitions

The algorithm mapCallT maps all transitions that enter procedure P. These transitions are passed to the algorithm as a set T. Only transitions from T with a push symbol equal to ε can be necessary to modify. Let $(q, a, pop, push, r)$ be a transition from T where $push = \varepsilon$, the $push$ symbol can be replaced with a call symbol from $stack(r)$ if and only if $|stack(r)| = 1$, meaning that state r has not been part of any procedure before. In other cases, the substitution is unnecessary as the transition $(q, a, pop, push, r)$ is already part of a larger procedure.

Algorithm 3: mapCallT

```
Input: NPDA $M = (Q^M, \Sigma, \Gamma_\varepsilon, \delta^M, I^M, \lambda^M, F^M, \phi^M)$,
       transitions $T \subseteq (Q^M \times \Sigma \times \Gamma_\varepsilon^M \times \Gamma_\varepsilon \times Q^M)$,
       $stack : Q^M \to 2^{\Gamma_\varepsilon^M}$ and $procS : Q^M \to Q^M$
Output: NPDA $N = (Q, \Sigma, \Gamma_\varepsilon, \delta, I, \lambda, F, \phi)$, such that
        $L(N) \equiv L(M)$
1 $N \leftarrow M$
2 forall $(q, a, \alpha, \beta, r) \in T$ do
3     if $\alpha = \varepsilon \wedge \beta = \varepsilon \wedge |stack(q)| = 1$ then
4         $\beta \leftarrow unpack(stack(r))$
5     else if $\alpha \neq \varepsilon \wedge \beta = \varepsilon$ then
6         $\beta \leftarrow unpack(stack(r))$        // $|stack(r)| = 1$
7     $\delta(q, a, \alpha) \leftarrow \delta(q, a, \alpha) \cup \{(procS(r), \beta)\}$
8 return $N$
```

Return Transitions

Algorithm 4: mapRetT

```
Input: NPDA $M = (Q^M, \Sigma, \Gamma_\varepsilon^M, \delta^M, I^M, \lambda^M, F^M, \phi^M)$,
       transitions $T \subseteq (Q^M \times \Sigma \times \Gamma_\varepsilon^M \times \Gamma_\varepsilon^M \times Q^M)$,
       $stack : Q^M \to 2^{\Gamma_\varepsilon^M}$ and $procS : Q^M \to Q^M$
Output: NPDA $N = (Q, \Sigma, \Gamma_\varepsilon, \delta, I, \lambda, F, \phi)$, such that
        $L(N) \equiv L(M)$
1  $N \leftarrow M$
2  forall $(q, a, \alpha, \beta, r) \in T$ do
3      if $\alpha = \varepsilon \wedge \beta = \varepsilon \wedge |stack(q)| \neq 1$ then
4          forall $\alpha \in stack(q)$ do
5              $\delta(procS(q), a, \varepsilon) \leftarrow \delta(procS(q), a, \alpha)$
                                   $\cup \{(r, \alpha)\}$
6          continue
7      if $\alpha = \varepsilon \wedge \beta = \varepsilon \wedge |stack(q)| = 1$ then
8          $\alpha \leftarrow unpack(stack(q))$
9      else if $\alpha = \varepsilon \wedge \beta \neq \varepsilon$ then
10         $\alpha \leftarrow unpack(stack(q))$        // $|stack(q)| = 1$
11     $\delta(procS(q), a, \alpha) \leftarrow \delta(procS(q), a, \alpha) \cup \{(r, \beta)\}$
12 return $N$
```

The mapping of return transitions that exit a procedure P is analogous to the mapping of call transitions. The transitions are passed to the algorithm as a set T. Only transitions from T with a pop symbol equal to ε

may require modification. The modification is similar to that described in Algorithm 3. However, if the transition $(q,a,\varepsilon,push,r)$ occurs and $|stack(q)| > 1$, then the transition $(q,a,\alpha,push,r)$ must be mapped to the procedure for each $\alpha \in stack(q)$.

Procedure Transitions

The procedure transitions are passed to the algorithm as $T \subseteq ((Q^M \times Q^M) \times \Sigma \times \Gamma_\varepsilon^M \times \Gamma_\varepsilon^M \times (Q^M \times Q^M))$. This set is transformed into $T' \subseteq (Q^M \times \Sigma \times \Gamma_\varepsilon^M \times \Gamma_\varepsilon^M \times Q^M)$ through the following process: the transition $((r,r'),a,\varepsilon,\varepsilon,(s,s'))$ is transformed to $(r,a,\varepsilon,\varepsilon,s)$ if a transition $(s,\varepsilon) \in \delta^M(r,a,\varepsilon)$ and an equivalent transition $(s',\varepsilon) \in \delta^M(r',a,\varepsilon)$ exist. If the transition is specific to states r and s, it is transformed to (r,a,α,α,s), for every $\alpha \in stack(r)$. Similarly, if the transition is specific to states r' and s'.

```
Algorithm 5: mapProcT
   Input: NPDA $M = (Q^M,\Sigma,\Gamma_\varepsilon^M,\delta^M,I^M,\lambda^M,F^M,\phi^M)$,
          transitions
          $T \subseteq ((Q^M \times Q^M) \times \Sigma \times \Gamma_\varepsilon^M \times \Gamma_\varepsilon^M \times (Q^M \times Q^M))$,
          $stack : Q^M \to 2^{\Gamma_\varepsilon^M}$ and $procS : Q^M \to Q^M$
   Output: NPDA $N = (Q,\Sigma,\Gamma_\varepsilon,\delta,I,\lambda,F,\phi)$, such that
          $L(N) \equiv L(M)$
   /* Transform $T$ to $(Q^M \times \Sigma \times \Gamma_\varepsilon^M \times \Gamma_\varepsilon^M \times Q^M)$
      format */
 1 $T' \leftarrow \emptyset$
 2 forall $((r,r'),a,\alpha,\beta,(s,s')) \in T$ do
 3     if $\alpha = \varepsilon \wedge \beta = \varepsilon$ then
 4         if $a \in \sigma_\varepsilon(r,s) \cap \sigma_\varepsilon(r',s')$ then
 5             $T' \leftarrow T' \cup \{(r,a,\alpha,\beta,s)\}$
 6         else if $a \in \sigma_\varepsilon(r,s) \setminus \sigma_\varepsilon(r',s')$ then
 7             forall $\eta \in stack(r)$ do
 8                 $T' \leftarrow T' \cup \{(r,a,\eta,\eta,s)\}$
 9         else
10             forall $\eta \in stack(r')$ do
11                 $T' \leftarrow T' \cup \{(r,a,\eta,\eta,s)\}$
12     else
13         $T' \leftarrow T' \cup \{(r,a,\alpha,\beta,s)\}$
14 $N \leftarrow M$
15 forall $(q,a,\alpha,\beta,r) \in T'$ do
16     $\delta(procS(q),a,\alpha) \leftarrow \delta(procS(q),a,\alpha)$
                                    $\cup \{(procS(r),\beta)\}$
17 return $N$
```

Auxiliary Transitions

All auxiliary transitions between two states of the procedure P that are not classified as procedure transitions are passed to the algorithm as a set T. Each transition in T is modified by substituting ε symbols with the corresponding stack symbols. Consider a transition (q,a,α,β,r). If $\alpha = \varepsilon$, it is substituted with a symbol from $stack(q)$. Similarly, if $\beta = \varepsilon$, it is substituted with a symbol from $stack(r)$.

```
Algorithm 6: mapAuxT
   Input: NPDA $M = (Q^M,\Sigma,\Gamma_\varepsilon^M,\delta^M,I^M,\lambda^M,F^M,\phi^M)$,
          transitions $T \subseteq (Q^M \times \Sigma \times \Gamma_\varepsilon^M \times \Gamma_\varepsilon^M \times Q^M)$,
          $stack : Q^M \to 2^{\Gamma_\varepsilon^M}$ and $procS : Q^M \to Q^M$
   Output: NPDA $N = (Q,\Sigma,\Gamma_\varepsilon,\delta,I,\lambda,F,\phi)$, such that
          $L(N) \equiv L(M)$
 1 $N \leftarrow M$
 2 forall $(q,a,\alpha,\beta,r) \in T$ do
 3     if $\alpha = \varepsilon \wedge \beta = \varepsilon$ then
 4         $\alpha \leftarrow unpack(stack(q))$      // $|stack(q)| = 1$
 5         $\beta \leftarrow unpack(stack(r))$       // $|stack(r)| = 1$
 6     else if $\alpha = \varepsilon \wedge \beta \neq \varepsilon$ then
 7         $\alpha \leftarrow unpack(stack(q))$      // $|stack(q)| = 1$
 8     else if $\alpha \neq \varepsilon \wedge \beta = \varepsilon$ then
 9         $\beta \leftarrow unpack(stack(r))$       // $|stack(r)| = 1$
10     $\delta(procS(q),a,\alpha) \leftarrow \delta(procS(q),a,\alpha)$
                                    $\cup \{(procS(r),\beta)\}$
11 return $N$
```

Initial States

The mapping of initial states into a procedure P is crucial as it determines how the resulting NPDA nondeterministically puts the initial symbols from Γ_ε onto the stack according to the function $\lambda : I \to 2^{\Gamma_\varepsilon}$. If the original initial state p is mapped into the procedure state s, then the s will become an initial state and will nondeterministically put the symbol from $stack(p)$ onto the stack.

```
Algorithm 7: mapInitS
   Input: NPDA $M = (Q^M,\Sigma,\Gamma_\varepsilon^M,\delta^M,I^M,\lambda^M,F^M,\phi^M)$,
          procedure $P = (V_P,E_P)$, and two stack symbols
          $stackU, stackV \in \Gamma_\varepsilon^M$
   Output: NPDA $N = (Q,\Sigma,\Gamma_\varepsilon,\delta,I,\lambda,F,\phi)$, such that
          $L(N) \equiv L(M)$
 1 $N \leftarrow M$
 2 forall $(u,v) \in V_P$ do
 3     $initStackU \leftarrow \emptyset$
 4     $initStackV \leftarrow \emptyset$
 5     if $u \in I$ then
 6         $initStackU \leftarrow \phi(u)$ if $\phi(u) \neq \{\varepsilon\}$ else $stackU$
 7     if $v \in I$ then
 8         $initStackV \leftarrow \phi(v)$ if $\phi(v) \neq \{\varepsilon\}$ else $stackV$
 9     if $initStackU \cup initStackV \neq \emptyset$ then
10         $I \leftarrow I \cup \{uv\}$   /* procedure state */
11         $\lambda(uv) \leftarrow initStackU \cup initStackV$
12 return $N$
```

Final States

The mapping of final states into a procedure P is important as it determines when the NPDA accepts the input (i.e., which symbol must be at the top of the stack) according to the function $\phi : F \to 2^{\Gamma_\varepsilon}$. If the original final state p is mapped into the procedure state

s, then the s becomes a final state that accepts input if the symbol from $stack(p)$ is at the top of the stack.

Algorithm 8: mapFinS

```
Input: NPDA $M = (Q^M, \Sigma, \Gamma_\varepsilon^M, \delta^M, I^M, \lambda^M, F^M, \phi^M)$,
       procedure $P = (V_P, E_P)$, and two stack symbols
       $stackU, stackV \in \Gamma_\varepsilon^M$
Output: NPDA $N = (Q, \Sigma, \Gamma_\varepsilon, \delta, I, \lambda, F, \phi)$, such that
        $L(N) \equiv L(M)$
1  $N \leftarrow M$
2  forall $(u, v) \in V_P$ do
3      $finStackU \leftarrow \emptyset$
4      $finStackV \leftarrow \emptyset$
5      if $u \in I$ then
6          $finStackU \leftarrow \phi(u)$ if $\phi(u) \neq \{\varepsilon\}$ else $stackU$
7      if $v \in I$ then
8          $finStackV \leftarrow \phi(v)$ if $\phi(v) \neq \{\varepsilon\}$ else $stackV$
9      if $finStackU \cup finStackV \neq \emptyset$ then
10         $F \leftarrow F \cup \{uv\}$ /* procedure state */
11         $\phi(uv) \leftarrow finStackU \cup finStackV$
12 return $N$
```

5. Experimental Results

The reduction algorithm that transforms NFA into the equivalent NPDA was evaluated on automata from the abstract regular model checking study [5], small regular expressions from the Snort database of network intrusion detection system[2], and large regular expressions describing protocols and attacks obtained from the L7 classifier for the Linux Netfilter framework[3] and Snort.

Prior to using the reduction approach, the input automata were simplified using the RABIT tool[4], which uses state merging and transition pruning based on language inclusions. For large regular expressions describing protocols and attacks containing up to millions of transitions, the approximation reduction method [7] was applied before using RABIT.

Our approach significantly reduced the results of the RABIT tool, with a reduction of 46% in the number of states and 13.6% in the number of transitions for automata from regular model checking, 24.5% in states and 16.3% in transitions for small regular expressions, and 43.9% in states and 36.4% in transitions for large regular expressions.

5.1 Regular Model Checking

The effectiveness of the algorithm was evaluated on 208 automata obtained from regular model checking, with up to 6800 transitions and 1600 states. The reduce tool RABIT was first used on the automata to merge language equivalent states and eliminate redundant transitions, after which our reduction algorithm was applied. The improvement in reduction was analyzed by comparing the results with the outputs of RABIT. The following graphs demonstrate the difference in the reduction of the number of states and transitions (which is the most significant measurement as noted in Subsection 4.1) in the automata.

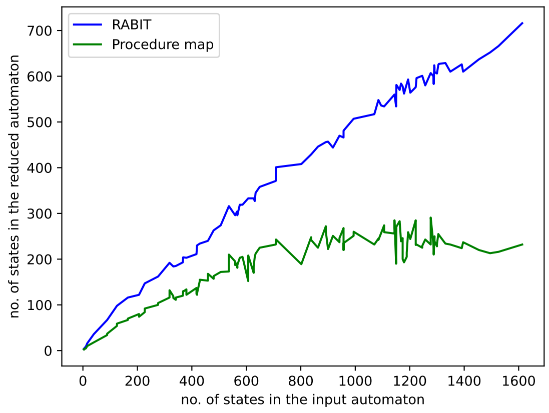


Figure 5. The graph presents a comparison of the number of states in the automata when the reduction tool RABIT is used alone or in the combination with the procedure mapping algorithm. The use of the procedure mapping algorithm resulted in a 58% decrease in the number of states compared to the results obtained from RABIT alone.

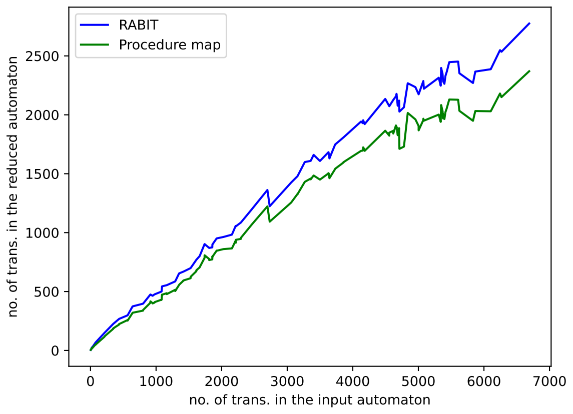


Figure 6. The graph demonstrates a comparison of the number of transitions in the automata when the reduction tool RABIT is used individually or in conjunction with the procedure mapping algorithm. The use of the procedure mapping algorithm resulted in a 14.6% reduction in the number of transitions compared to the results obtained from the standalone use of RABIT.

[2]Available at: `http://snort.org`
[3]Available at: `http://netfilter.org`
[4]Available at: `http://languageinclusion.org`

5.2 Regular Expressions

The procedure mapping reduction is found to be more effective compared to state merging and transition pruning when applied to automata representing regular expressions. These automata often have a tree-like or even linear structure, resulting in a limited number of language equivalent states.

The reduction was tested on 656 nearly linear automata obtained from Snort, with a maximum of 410 states and 410 transitions. The following graphs demonstrate the comparison between the reduction achieved by the RABIT tool and the reduction achieved by the procedure mapping algorithm. It is evident that the RABIT tool was unable to produce a meaningful reduction.

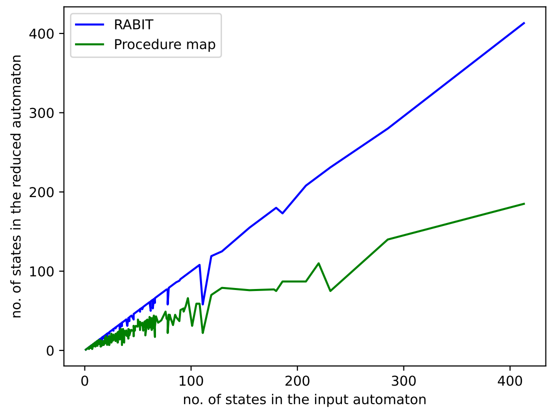


Figure 7. The reduction tool RABIT failed to achieve meaningful reduction on the input automata. In contrast, the procedure mapping reduction resulted in an average reduction of 25.9% in the number of states of the input automata.

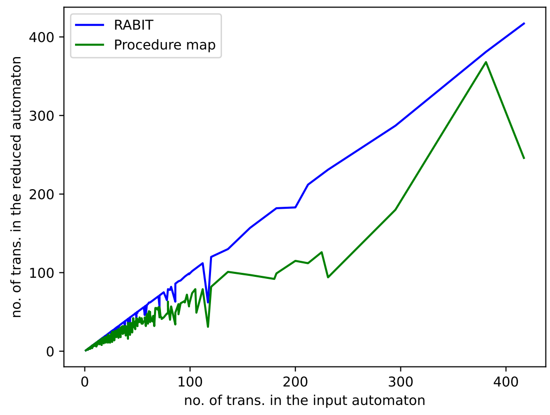


Figure 8. The reduction tool RABIT was unsuccessful in reduction on the input automata. On the other hand, the procedure mapping reduction was able to achieve an average reduction of 16.3% in the number of transitions of the input automata.

5.3 Protocols and Attacks

Last but certainly not least, the procedure mapping reduction was tested on large automata representing regular expressions for protocols and attacks obtained from the L7 classifier for the Linux Netfilter framework and the Snort tool. The tested automata, `backdoor`, `pop3`, and `spyware`, were obtained from the Snort tool and provide a description of attacks on selected protocols. The `l7-all` automaton was obtained from the L7 classifier and describes attacks on a selected set of protocols.

Automaton	States	Transitions
`backdoor`	3,898	100,301
`pop3`	923	209,467
`spyware`	12,809	279,334
`l7-all`	7,280	2,647,620

Table 1. The sizes of the original automata representing regular expressions for protocols and attacks.

As the automata contain hundreds of thousands, or even millions, of transitions, reducing their size is crucial before applying the RABIT tool or procedure mapping reduction. To achieve this, the approximation reduction (as described in [7]) is used. This reduction method involves a reduction in the size of the automaton at the cost of a loss in language precision. The greater the reduction ratio achieved through approximation reduction, the lower the precision.

Automaton	Red. Ratio	Precision	Algorithm	States
`backdoor`	30%	100%	Approx. Red.	1,169
			RABIT	690
			RABIT + Proc.	375
`l7-all`	30%	35%	Approx. Red.	2,184
			RABIT	210
			RABIT + Proc.	133
`pop3`	30%	N/A	Approx. Red.	277
			RABIT	77
			RABIT + Proc.	36
`spyware`	30%	100%	Approx. Red.	3,843
			RABIT	650
			RABIT + Proc.	345
	15%	35%	Approx. Red.	1,921
			RABIT	244
			RABIT + Proc.	154

Table 2. The table presents the results of the reduction for automata that have been pre-processed using the approximation reduction method. It displays the difference in the number of states in the resulting automata when the RABIT reduction tool is used alone or in the combination with the procedure mapping algorithm.

After the approximation reduction, the RABIT tool was used to merge language equivalent states and remove redundant transitions, followed by the application of the procedure mapping reduction. The improvement in reduction was evaluated by comparing the results with the output of RABIT. The results obtained for different approximation reduction ratios are shown in the tables 2 and 3.

Automaton	Red. Ratio	Precision	Algorithm	Transitions
`backdoor`	30%	100%	Approx. Red.	31,080
			RABIT	10,900
			RABIT + Proc.	7,600
`l7-all`	30%	35%	Approx. Red.	1,230,859
			RABIT	9,828
			RABIT + Proc.	6,997
`pop3`	30%	N/A	Approx. Red.	46,523
			RABIT	8,578
			RABIT + Proc.	4,226
`spyware`	30%	100%	Approx. Red.	177,302
			RABIT	14,576
			RABIT + Proc.	9,276
	15%	35%	Approx. Red.	107,970
			RABIT	8,408
			RABIT + Proc.	6,395

Table 3. The table presents the reduction outcomes for automata that have been pre-processed with the approximation reduction method. It displays the differences in the number of transitions in the resulting automata when using the RABIT reduction tool solely or in conjunction with the procedure mapping algorithm.

The results from the tables show that the procedure mapping reduction further reduces the number of states and transitions in the resulting RABIT automaton. On average, the procedure mapping algorithm outputs 43.9% fewer states and 36.4% fewer transitions. The greatest reduction in states was 53.2% for the `pop3` automaton, and the greatest reduction in transitions was 50.7% for the same automaton.

This experiment concluded that the procedure mapping reduction is the most efficient for the automata representing regular expressions.

6. Conclusion

In this paper, a new reduction approach for NFAs is presented. Traditional minimization techniques, such as state merging and transition pruning, often result in NFAs that still contain redundant transition sequences. Our reduction algorithm replaces those redundant transition sequences in the NFA with a single procedure by transforming the input NFA into an equivalent NPDA. The information about the original branch of the automaton, entry points, and return locations is stored on the stack. This transformation results in a more efficient representation of the input NFA. Additionally, the language generated by the resulting NPDA is regular due to the limited size of the stack, which is at most one symbol.

Our reduction algorithm was evaluated on a diverse set of automata, including 208 automata obtained from regular model checking, 656 nearly linear automata representing regular expressions obtained from Snort, and automata representing large regular expressions describing protocols and attacks obtained from the L7 classifier for the Linux Netfilter framework and the Snort tool.

The results of our approach showed a significant reduction compared to the RABIT tool, with reductions of 46% in the number of states and 13.6% in the number of transitions for automata from regular model checking, 24.5% in states and 16.3% in transitions for small regular expressions, and 43.9% in states and 36.4% in transitions for large regular expressions. The greatest reduction in states was 53.2% for the `pop3` automaton representing regular expressions for protocols and attacks, and the greatest reduction in transitions was 50.7% for the same automaton.

The experiments show that the minimization technique using procedure mapping is the most suitable for automata representing regular expressions due to their tree-like or almost linear structure.

7. Future Work

In future research, we aim to optimize the reduction algorithm by defining a heuristic function to select the best procedure candidate based on the number of possible return transitions. The next optimization step is to substitute stack symbols to increase the number of transitions with identical pop and push symbols. A final improvement to the algorithm is adjusting the start and end points of the procedure to reduce the number of return and call transitions.

8. Acknowledgment

I would like to express my gratitude to my supervisor, Mgr. Lukáš Holík Ph.D, for his support and the time he devoted to my work on this research.

References

[1] Abdulla, A. P., Holík, L., Chen, Y.-F., Mayr, R. and Vojnar, T. *When Simulation Meets Antichains (On Checking Language Inclusion of Nondeterministic Finite (Tree) Automata).* 2010. 22 p. Available at: `https://www.fit.vut.cz/research/publication/9152`.

[2] Abdulla, P. A., Atig, M. F., Chen, Y.-F., Holík, L., Rezine, A. et al. String Constraints for Verification. In: Biere, A. and Bloem, R., ed. *Computer Aided Verification*. Cham: Springer International Publishing, 2014, p. 150–166. ISBN 978-3-319-08867-9.

[3] Ain, Q., Saeed, Y., Naseem, S., Ahamd, F., Alyas, T. et al. DNA Pattern Analysis using Finite Automata. *International Research Journal of Computer Science (IRJCS)*. october 2014, vol. 1, p. 1–4.

[4] Aziz, A., Singhal, V., Brayton, R. and Swamy, G. Minimizing interacting finite state machines: a compositional approach to language containment. In: *Proceedings 1994 IEEE International Conference on Computer Design: VLSI in Computers and Processors*. 1994, p. 255–261.

[5] Bouajjani, A., Habermehl, P., Rogalewicz, A. and Vojnar, T. Abstract regular (tree) model checking. *International Journal on Software Tools for Technology Transfer*. Apr 2012, vol. 14, no. 2, p. 167–191. Available at: https://doi.org/10.1007/s10009-011-0205-y. ISSN 1433-2787.

[6] Bustan, D. and Grumberg, O. Simulation Based Minimization. In: McAllester, D., ed. *Automated Deduction - CADE-17*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, p. 255–270. ISBN 978-3-540-45101-3.

[7] Češka, M., Havlena, V., Holík, L., Korenek, J., Lengál, O. et al. Deep Packet Inspection in FPGAs via Approximate Nondeterministic Automata. In: *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 2019, p. 109–117.

[8] Clemente, L. and Mayr, R. Efficient reduction of nondeterministic automata with application to language inclusion testing. *CoRR*. 2017, abs/1711.09946. Available at: http://arxiv.org/abs/1711.09946.

[9] Fiedor, T., Holík, L., Lengál, O. and Vojnar, T. Nested antichains for WS1S. *Acta Informatica*. 2019, vol. 56, no. 3, p. 205–228. Available at: https://doi.org/10.1007/s00236-018-0331-z.

[10] Fu, C., Deng, Y., Jansen, D. and Zhang, L. On Equivalence Checking of Nondeterministic Finite Automata. In: Larsen, K. G., Sokolsky, O. and Wang, J., ed. *Dependable Software Engineering. Theories, Tools, and Applications*. Cham: Springer International Publishing, 2017, p. 216–231. ISBN 978-3-319-69483-2.

[11] Henzinger, M., Henzinger, T. and Kopke, P. Computing simulations on finite and infinite graphs. In: *Proceedings of IEEE 36th Annual Foundations of Computer Science*. 1995, p. 453–462.

[12] Ilie, L., Navarro, G. and Yu, S. On NFA Reductions. In: Karhumäki, J., Maurer, H., Păun, G. and Rozenberg, G., ed. *Theory Is Forever: Essays Dedicated to Arto Salomaa on the Occasion of His 70th Birthday*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, p. 112–124. Available at: https://doi.org/10.1007/978-3-540-27812-2_11. ISBN 978-3-540-27812-2.

[13] Rabin, M. and Scott, D. Finite Automata and Their Decision Problems. *IBM Journal of Research and Development*. april 1959, vol. 3, p. 114–125.

[14] Sourdis, I. and Pnevmatikatos, D. Fast, Large-Scale String Match for a 10Gbps FPGA-Based Network Intrusion Detection System. In: Y. K. Cheung, P. and Constantinides, G. A., ed. *Field Programmable Logic and Application*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, p. 880–889. ISBN 978-3-540-45234-8.