DOI: 10.15514/ISPRAS-2025-37(6)-33

Bisimulations in Memory Finite Automata

[1] *A.N. Nepeivoda, ORCID:* 0000-0003-3949-2164 *<a_nevod@mail.ru>*
[2] *A.D. Delman, ORCID: 0009-0009-6885-8429 <adelman2112@gmail.com>*
[2] *A.S. Terentyeva, ORCID: 0009-0006-8547-3959 <mathhyyn@gmail.com>*

[1] *Ajlamazyan Program Systems Institute of the Russian Academy of Sciences, 4a, Piotr 1 st., Veskovo, Yaroslavskaya obl., 152024, Russia.*

[2] *Bauman Moscow State Technical University, 5-1, 2nd Baumanskaya st., Moscow, 105005, Russia.*

Abstract. This research aims at studying the bisimulation relation for memory finite automata, which are used as the automata model for extended regular expressions in the series of works, and encapsulate the expressiveness of the named capture groups. We propose an experimental algorithm for checking bisimulation of one-memory MFAs. For multi-memory automata, we show that, in some borderline cases, the bisimulation problem is closely related to a question of whether a parameterized word is always a solution of a given word equation of an arbitrary form.

Keywords: extended regular expressions; memory finite automata; bisimulation; word equations

For citation: Nepeivoda A.N., Delman A.D., Terentyeva A.S. Bisimulations in Memory Finite Automata. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 6, part 3, 2025, pp. 7-18. DOI: 10.15514/ISPRAS-2025-37(6)-33.

Acknowledgements. The first author was partially supported by Russian Academy of Sciences, research project No.125021302067-9.

Бисимуляции конечных автоматов с памятью

[1] *А.Н. Непейвода, ORCID:* 0000-0003-3949-2164 *<a_nevod@mail.ru>*
[2] *А.Д. Дельман, ORCID: 0009-0009-6885-8429 <adelman2112@gmail.com>*
[2] *А.С. Терентьева, ORCID: 0009-0006-8547-3959 <mathhyyn@gmail.com>*
[1] *Институт программных систем им. А.К. Айламазяна РАН,*
Россия, 152024, Ярославская обл., с. Веськово, ул. Петра Первого, д. 4а.
[2] *Московский государственный технический университет имени Н.Э. Баумана,*
Россия, 105005, Москва, 2-я Бауманская улица, д. 5. стр. 1

Аннотация. В статье рассматривается проблема бисимуляции автоматов с памятью, представляющих собой модель расширенных регулярных выражений с группами захвата в память. Построен алгоритм бисимуляции таких автоматов в случае единственной ячейки памяти. Показано, что в случае нескольких ячеек и возможности рекурсивного перезахвата в память, проблема бисимуляции включает в себя проблему проверки истинности произвольного уравнения в словах над элементами линейных контекстно-свободных языков.

Ключевые слова: расширенные регулярные выражения; конечные автоматы с памятью; бисимуляция; уравнения в словах.

Для цитирования: Непейвода А.Н., Дельман А.Д., Терентьева А.С. Бисимуляции конечных автоматов с памятью. Труды ИСП РАН, том 37, вып. 6, часть 3, 2025 г., стр. 7–18 (на английском языке). DOI: 10.15514/ISPRAS–2025–37(6)–33.

Благодарности: Первый автор поддержан проектом НИР РАН, номер проекта 125021302067-9.

1. Introduction

Extended regular expressions have been known at least since the early 90s, when they were implemented in the text editor ed [1]. Many practical extensions of the regexes are made inside the class of regular languages, e.g., lookaheads and lookbehinds. The main exception is a back-reference support: if a capture group contains an iterated expression fragment, then back-referencing to the group may represent non-regular properties of the recognized language. For example, the expression $(a*)(b*)\backslash 1\backslash 2$ recognizes the language $\{a^n b^m a^n b^m \mid m,n \in \mathbb{N}\}$, which is a typical example of a non-context free language.

The main concern about the extended regex models is the high computational complexity of their analysis. The language inclusion problem for extended regexes even with a single memory cell is proved to be undecidable [2], the similar statement holds for language inclusion for patterns that are modelled with extended regular expressions with no restriction on memorized values and no loops and alternations in the core regex [3]. Thus, extended regex simplification tends to be a hard problem, requiring the development of approximate solutions.

It is known that some practical tools such as RE2 [4] process even academic regular expressions via non-deterministic finite automata optimizations, because they can be much faster than the exact minimization algorithm, and can preserve the structure of the regular expressions. One of such NFA optimization algorithms is merging the bisimilar state classes [5-6], which is also a well-known technique in program optimization. If some of the states in an NFA are indistinguishable from the point of view of a user of the NFA, these states can be considered as a single state, thus reducing the state space with no impact on computation traces. Equivalence of NFA, which is known to be in EXPTIME, can be tested via bisimulation as well [7]: although the bisimulation relation is finer than the language equivalence, it can be computed in polynomial time, thus giving a fast under-approximation of the equivalence test. In the case of pushdown automata, language equivalence is undecidable, while bisimulation is decidable (but non-elementary) [8-9]. Bisimulation was also applied to symbolic finite automata, i.e., finite automata with guarded transitions, in order to improve performance of extended regexes with no memory operations [10].

It seems very natural to consider bisimulation-based optimizations in the presence of capture groups, both because the bisimilarity is typically easier to compute than the language equivalence, and because the bisimulation-merging optimizations are structure-preserving, which is practical in cases when the captured data is used outside the extended regex. However, as far as we know, none of state machine formalisms supporting backreferences were considered in the papers studying the bisimulation-based optimizations and analysis. The main reason for this gap is maybe a confusion of different backreference-based formalisms for extended regular expressions, none of which is chosen as a standard nowadays, despite the fact that sometimes distinctions are minor, and some formalisms can be treated as special cases of the others [11].

It is also worth noting that the language of backref-regexes cannot be treated as a special case of a formal language with well-known bisimulation properties. It can be easily shown that $\{a^n b^n | n \in \mathbb{N}\}$, which is both context-free and Petri net language, cannot be recognized by any backref-regex. On the other hand, the language $\{\omega\omega \mid \omega \in (a|b)^*\}$ is trivially captured by the regex $\big((a|b)^*\big)\backslash 1$, while this language is known to be neither context-free nor Petri-net [12]. Thus, the backref-regexes formalism is independent from the classification of process algebras given in the paper [13].

This research aims at studying the bisimulation relation for the memory finite automata (MFA), which are used as the automata model of the extended regular expressions in the series of works [2, 14], and encapsulate the expressiveness of the named capture groups with reinitializations. We propose an experimental algorithm for checking bisimulation of one-memory MFAs, and discuss why bisimulation is hard in the case of multiple memory cells. For the latter, we show that, in some borderline cases, the bisimulation problem is closely related to a question whether a parameterized is always a solution to a given word equation of an arbitrary form.

2. Preliminaries

2.1 Bisimulation

Every state machine can be defined by its transition graph, which contains a complete description of its possible traces. If the state machine is not finite, the transition graph is infinite, taking into account the infinite set of inner states of the machine. We assume that the transition graphs are represented as labelled transition systems, in which edges are labelled by the actions possible in the state machines.

Definition 2.1.1 Given labelled transition systems τ_1, τ_2 and the action alphabet Σ, *bisimulation* is a coarsest relation ~ between states of the systems satisfying the following property.

If $q_1 \in States(\tau_1)$, $q_2 \in States(\tau_2)$, $\gamma \in \Sigma$, and $q_1 \sim q_2$, then for every transition $q_1 \xrightarrow{\gamma} {q'}_1 \in \tau_1$ there is a transition $q_2 \xrightarrow{\gamma} {q'}_2 \in \tau_2$ s.t. $\tau_1 \sim \tau_2$, and vice versa.

Systems τ_1 and τ_2 are bisimilar if and only if their starting states are in bisimulation, and, in the case of the existence of final states, any final state in τ_1 is bisimilar to a final state in τ_2, and vice versa.

State machines are bisimilar if and only if their transition graphs are bisimilar.

For most known state machine models, the bisimulation relation is strictly finer than the language equivalence relation. E.g., non-bisimilar finite automata recognizing the same language $\{a^n + 1 | \, n \in \mathbb{N}\}$ are given in Fig. 1.

A simple and natural technique for checking the bisimilarity of the transition graphs τ_1 and τ_2 can be formulated as a bisimilarity game with two players:

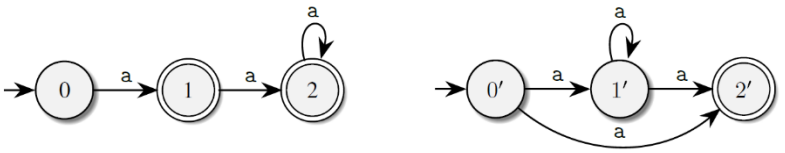
The initial player configuration is the pair $\langle q_S, {q'}_S \rangle$, where q_S, ${q'}_S$ are the starting states of τ_1 and τ_2 respectively.

Given the pair $\left\langle q_{k_i}, q_{k_j} \right\rangle$, Attacker chooses any element of the pair and a transition $q_{k_m} \xrightarrow{\gamma} q_p$ in the corresponding LTS. Defender must respond with a transition $q_{k_n} \xrightarrow{\gamma} {q'}_p$ from the remaining state in

the pair, respecting the finality of the state q_p. After this round, the player configuration includes q_p and q'_p, and the new round starts.

Attacker can play while there exists at least one transition from either q_{k_i} or q_{k_j}. If Defender cannot choose a transition at least for one player configuration reachable from the initial configuration, the LTS are not bisimilar. Otherwise, the LTS are bisimilar.

E.g., given the processes in Fig. 1, Attacker can choose the first one (namely, A_1) and play $0 \xrightarrow{a} 1$. Then, in order to respect the state finality, Defender is forced to respond with $0' \xrightarrow{a} 2'$. Since state $2'$ has no outgoing transitions, any possible second action of Attacker in the LTS makes Defender to lose.



(a) Glushkov NFA $\mathscr{A}_1$ for aa*. (b) Glushkov NFA $\mathscr{A}_2$ for a*a.

Fig. 1. An example of non-bisimilar equivalent NFA.

2.2 Extended Regular Expressions

The theory of the extended regular expressions followed much later than they became usual in practice. Several formalisms were proposed by different research groups, based on the details of naming capture groups and possibility of reinitialization of the groups [11]. In 2014, Markus Schmid suggested to consider only named capture groups in the extended syntax, and proposed a convenient representation in terms of state machines with restricted memory support (memory finite automata, MFA). Schmid-style regular expressions are more expressive than PCRE2-style regular expressions used in practice currently [11], but the trends of the PCRE2 development show that cyclic re-initializations are likely to appear in near future [15].

In our work, we use the latest MFA model, which includes reset memory actions [16]. Following the papers [14, 16], we also call extended regular expressions ref-words.

Definition 2.2.1 Given an input alphabet Σ and the memory set cardinality $k \in \mathbb{N}$, *a regular expression with backreferences (ref-word)* is defined recursively:

- $\gamma \in \Sigma$, ϵ, and $\&i$, where $i \leq k$, are ref-words (the latter defines reading the □-th memory cell);
- if ρ_1 and ρ_2 are ref-words, then so are $(\rho_1|\rho_2)$, $(\rho_1\rho_2)$, (ρ_1)*;
- if $i \leq k$ and ρ is a ref-word including neither $\&i$ nor $[_i\tau]_i$, then $[_i\tau]_i$ is also a ref-word.

The last operation defines *capture groups*. We require memory brackets $[_i$, $]_i$ to be balanced both wrt the regular parentheses, and wrt each other. That is the only distinction from the formalism given in paper [14], which admits unbalanced capture groups.

The ref-word definition above does not specify semantics of uninitialized backreferences, e.g. in $\&1a[_1b*]_1a\&1$. Following the terminology of the paper [11], we assume *ϵ-semantics*: all uninitialized references are meant to have the empty value. Thus, the ref-word given above recognizes the language $\{ab^nab^n \mid n \in \mathbb{N}\}$.

2.3 Memory Finite Automata

Definition 2.3.1 A *memory finite automaton* (MFA) [14] is a tuple $\langle Q, \Sigma, q_0, F, \delta\rangle$, where Q is a finite set of states, Σ is the input alphabet, $q_0 \in Q$ is a starting state, $F \subseteq Q$ are final states, and $\delta: (Q \times \Sigma \cup \{\epsilon\} \cup \{1,2,..,k\}) \to Q \times \langle o, c, r, s\rangle^k$ is a transition table. The symbols o, c, r, s are

memory instructions for opening, closing, resetting, and staying in the current memory status respectively.

An MFA *configuration* is a tuple $\langle q, \omega, (u_1, m_1), .., (u_k, m_k)\rangle$, where q is a current state, ω is an input suffix to be read, and for all $i, 1 \le i \le k$, (u_i, m_i) is an i-th memory state, consisting of a stored string u_i and a memory status $m_i \in \{O, C\}$. Given an input ω_0, the initial memory state is $\langle q_0, \omega_0, (\epsilon, C), .., (\epsilon, C)\rangle$, i.e., all the memory cells are assumed to be closed and to store empty strings.

A transition from configuration $\langle q, \upsilon\omega, (u_1, m_1), .., (u_k, m_k)\rangle$ to $\langle q', \omega, (u'_1, m'_1), .., (u'_k, m'_k)\rangle$ is possible if there is a transition rule from q to q' labelled β, and having the instructions $\langle b_1, .., b_k\rangle$ such that either:

- $\beta \in \Sigma \cup \{\epsilon\}$ and $\upsilon = \beta$ (usual transition), or
- $\beta \in \{1, .., k\}$ and $\upsilon = u_\beta$, given $m'_\beta = C$ (reading transition).

The memory states are updated as follows:

$$m'_i = b_i(m_i) = \begin{cases} C, b_i = c \vee b_i = r \vee (b_i = s \wedge m_i = C) \\ O, b_i = o \vee (b_i = s \wedge m_i = O) \end{cases}, u'_i = \begin{cases} u_i, m'_i = C \\ \epsilon, b_i = r \\ \upsilon, b_i = o \\ u_i\upsilon, b_i = s \wedge m_i = O \end{cases}$$

That is, the memory status is updated before processing the tape. If the resulting memory status is closed, the memory cell does not append the currently read input fragment υ. If the status is open, the string υ is appended to the current memory. Both reset and open instructions clear the memory, but the first sets C as a status, while the second sets O.

An example of a memory finite automaton for a nonregular language $\{a^{n+k}ba^n | n > 0\}$ is shown in Fig. 2. For convenience, the memory actions on the edges are given in the brief form: a label only lists cells that are closed, reset and opened along the edge, not mentioning the staying in the current status instruction. The MFA that are used in the examples are uniformly generated from the corresponding ref-words, using an MFA construction algorithm based on the Glushkov construction [17], and utilising the reset memory action in order to avoid ϵ-transitions.

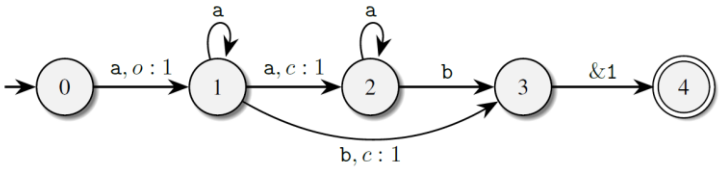


Fig. 2. An example of a memory finite automaton.

3. Bisimulation in Memory Finite Automata

Definition 3.1. Let $A = \langle Q, \Sigma, q_0, F, \delta\rangle$ be a memory finite automaton over k memory cells. Its *transition graph* $G(A)$ is defined as follows.

- $(q_0, \langle(\epsilon, C), .., (\epsilon, C)\rangle)$ is the starting node of the transition graph;
- given a configuration $N_i = (q, \langle(u_1, m_1), .., (u_k, m_k)\rangle)$, children of the node labelled with N_i are the nodes labelled with configurations reachable by all possible one-step transitions from N_i.

In most practical cases, the memory statuses in all states of A are determined by the states (i.e., given a state q, the values of $m_1, .., m_k$ can be restored independently of the previous trace). We assume that the condition holds for all MFA considered, and omit m_i values in the node configurations of transition graphs. While the notion "capture groups" is usually used in the context of ref-words, we also say that, given an MFA A the subgraphs between the open and close operations wrt the cell k

are "capture groups" for the reference k in A. Additionally, we assume that every capture group of an MFA is useful, i.e., there exists at least one path in the transition graph where the value accumulated in the group is used by a reference. In order to distinguish the actions with the same label in distinct A_1 and A_2, we sometimes mark the references in the transition graphs with the corresponding subscripts, i.e., $\&k_{A_1}$ and $\&k_{A_2}$.

Given the MFA formalism, we make the following general assumption about the input commands controlled by the user.

Assumption 3.1. Given the transition graphs $G(A_1)$ and $G(A_2)$, the transitions from nodes N_{j_1} and N_{j_2} both labelled with $\&i$ are equal if and only if the stored u_i value in the N_{j_1} configuration coincides with the stored u_i value in the configuration of N_{j_2} .

Now we are ready to give the definition of the bisimulation relation in the terms of MFA.

Definition 3.2. Given MFA A_1 and A_2, the MFA are *bisimilar* (denoted $A_1 \sim A_2$), if and only if their transition graphs $G(A_1)$ and $G(A_2)$ are bisimilar.

When considering the problem of MFA bisimulation, it is natural to assume that users have no direct access to memory operations (open, close, reset). On the other hand, a reference to a memory is an explicit control action, which is distinct from any other user action. Thus, we introduce the notion of an action automaton, which describes possible sequences of

control actions in the process graph of a MFA.

Definition 3.3. Given k-cell MFA A, its *action NFA* $\pi_M(A)$ results from A by erasing memory operations from edge labels of A. The symbols $\&i$ $(1 \leq i \leq k)$ become elements of the input action alphabet Act for $\pi_M(A)$.

If MFA A_1 and A_2 are in the bisimulation relation, then the control traces of them must coincide, thus, $\pi_M(A_1) \sim \pi_M(A_2)$. In the latter case, we say that A_1 and A_2 are action-bisimilar. This relation induces a relation on the states of the MFA themselves, however, the action-bisimilarity is not enough to provide real MFA bisimilarity: in order to imitate the action $\&k_{A_1}$ played by Attacker, not only Defender must respond with the action $\&k_{A_2}$, but also, they must guarantee that the reference reads exactly the same value. Otherwise, the two actions $\&k$ cannot be considered as equal.

For example, let us consider the MFA given in Fig. 3. Their action NFA are trivially bisimilar, being equal, but Attacker has the following winning strategy.

- Play $S' \xrightarrow{a} 1'$.
- If Defender responds with $S \xrightarrow{a} 1$, then play $1' \xrightarrow{\&1} 3'$. Defender loses, being unable to reset memory value to ϵ.
- If Defender responds with $S \xrightarrow{a} 2$, then play $1' \xrightarrow{a} 2'$. Now Defender cannot accumulate anything in the capture group.

4. Bisimulation for One-Cell Memory Finite Automata

We start with the simplest class of memory finite automata, namely, automata using a single memory cell. In order to obtain a winning strategy, Defender must be able to repeat the Attacker's decisions made inside the capture groups. The two possible sorts of decisions are:

- given a loop inside a capture group, decide whether to continue iterations or to exit;
- given a captured branching by different $\gamma_1, \gamma_2 \in \Sigma$, choose the alternation branch.

Note that, given a branching without a loop in a capture group, we are still able to capture only a finite set of possible strings, provided that the MFA has the only memory cell. Hence, the second sort of the decisions, considered separately, is somewhat "weaker" than the first: memorization of alternations without loops can be modelled by finite state models. Hence, we are interested in the

decisive actions occurring only in capture groups containing loops. If the actions are considered outside the capture groups, we mention this fact explicitly.

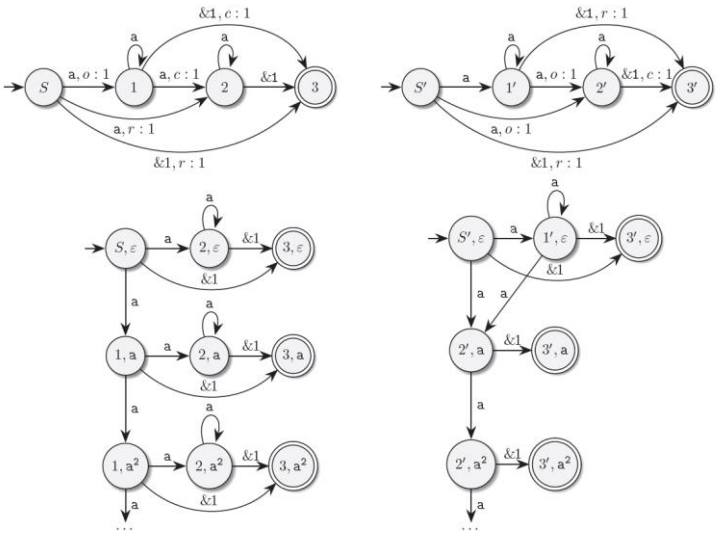


Fig. 3. Non-bisimilar MFA and their transition graphs.

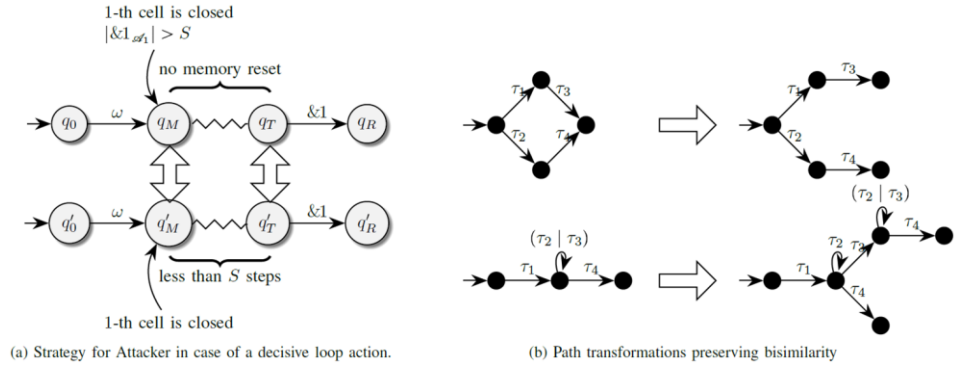
Proposition 4.1. Let $\pi_M(A_1) \sim \pi_M(A_2)$. If there exists a state q_1 in A_1 s.t. it has an outgoing decisive action (inside a capture group) and none of the states in A_2 that are action-bisimilar to q_1 has such a decisive action inside the capture group, then $\neg(A_1 \sim A_2)$.

Proof. Let us show Attacker's winning strategy in the case of the decisive action in A_1 which is not action-bisimilar to any decisive action in A_2. First, given the loop in the capture group where the decisive action occurred (or is to occur), Attacker decides to play the looping back action at least S times, where S is the sum of the number of states in A_1 and A_2, and additionally chooses such a number of iterations that the value of $\&1_{A_1}$ is not equal to $\&1_{A_2}$. Second, Attacker chooses the shortest possible path in A_1 to the action $\&1$. In order to imitate the action $\&1_{A_1}$, Defender is forced to reset the value of $\&1_{A_1}$, but the value of $\&1_{A_1}$ is longer than any possible value read along at most transitions. Due to MFA semantics, no transition captured in the memory cell 1 can refer to the memory cell, thus, no more than S letters can be captured. See Fig. 4 (a).

Proposition 4.1 demonstrates an important property of bisimilar traces within MFA semantics: if the memory cell captures any loop, then the capturing of the input fragment read along the loop must be done synchronously in both MFA. Otherwise, Attacker can apply the strategy shown in Fig.4 and force the Defender to lose, looping inside the capture long enough. Hence, in bisimilar MFA, all the traces with loops must be action-bisimilar in their "decisive" fragments, as well as synchronized in terms of write and read operations: if a trace of A_1 contains k read operations after a capturing with a loop, then the bisimilar trace of A_2 must also contain k read operations without re-capturing memory between them.

In Fig. 3, while the states 1 and $2'$ with the decisive actions are action-bisimilar, the process graphs are not. The reason is that Proposition 4.1 induces a narrowing on action-bisimilarity relation, taking into account the synchronisation that must occur in the capture groups previously visited. Namely, if the state 1 is visited by a player, then $2'$ is visited as well, and all the actions reachable after closure of the memory cell containing state 1 in its capture group must be also reachable after closure of the memory cell containing state $2'$. The given condition fails due to existence of state 2 in A_1, allowing Attacker to read additional letters a after the memory is closed, while in A_2 the only reachable state after the memory closure is the state $3'$, and Defender is unable to read any a in this state. Hence,

not only the decisive actions must be bisimilar in the bisimilar MFA, but also the sub-automata having the nodes with the decisive actions as starting states must be bisimilar.



(a) Strategy for Attacker in case of a decisive loop action.

(b) Path transformations preserving bisimilarity

Fig. 4. Attacker strategy in case of a decisive action in a loop within a capture group, and bisimulation-preserving path transformation.

Now we are to deal with the case when the memorized strings can be re-captured by Defender. According to Proposition 4.1, this case can only occur when the strings are read along the paths in that do not include any loops. The set of all possible memory values read along the paths is finite. Hence, we can annote the paths storing these values with a finite number of indexes, and then track that the indexes coincide when the memories are referenced to. One important case of such annotation is reset-annotation: that is, tracking that states on the given path store the empty value. However, in order to make the annotation, it is crucial to split the paths carrying unbounded and bounded strings in the memory cell. The following proposition is a corollary of the fact that ref-words satisfy certain Kleene algebra theorems, such as right distributivity, nesting, and fusion.

Proposition 4.2 Given any MFA A, we can transform it into a bisimilar MFA using the following transformations:

- given two paths having the same destination node, duplicate the destination node and split them apart (subfigure (b) in Fig. 4, upper diagram);
- and, given a looped path, extract a first iteration over a certain subpath into an explicit subgraph (subfigure (b) in Fig. 4, lower diagram).

Using these two transformations, we can split subpaths storing constant strings from the ones storing possibly unbounded strings, and apply the annotation procedure in order to validate constant value re-capturing.

For example, such transformations allow us to distinguish cases of bisimilar MFA based on the ref-words $a^*\big([_1 a^*]_1\big)^*\&1$ and $\big([_1 a^*]_1\big)^*\&1$ – where the latter is transformed to $\underbrace{a^*}_{\text{annotated with } \epsilon}\big([_1 a^*]_1\big)^*\&1$, and non-bisimilar MFA based on $[_1 b]_1 \underbrace{a^*}_{annotated\ with\ b}\big([_1 a^*]_1\big)^*\&1$ and $[_1 b]_1\big([_1 a^*]_1\big)^*\&1$. The bisimilarity in the second case is broken, because we cannot "unfuse" the looped capture group in $[_1 b]_1\big([_1 a^*]_1\big)^*\&1$ preserving the constant b-annotation. Hence, the path fragments preceding captured decisive actions cannot be made bisimilar by any combination of transformations given in Proposition 4.2.

It is tempting to conclude that the whole capture groups with loops are required to be action-bisimilar in order to guarantee that $A_1 \sim A_2$. Nevertheless, in some bisimilar MFA non-decisive actions can occur asynchronously even along the paths including loops. An example is given in Fig. 5. The capture groups are non-bisimilar, while the states 2 and $2'$ with action-decisive outgoing transitions are both captured. The reason for this non-trivial bisimulation is rooted in the theory of word equations. Namely, for every $\omega = a^k, a\omega = \omega a$, and that is why the values of the references $\&1_{A_1}$

and $\&1_{A_2}$ always coincide. Due to Proposition 4.1, non-bisimilarity of this sort, given a loop in the capture group, can occur only with respect to constant prefixes and suffixes surrounding the input fragment captured in the loop. That is, given the captured strings X_1, X_2, …, X_n, they can be rearranged between constant fragments in such a way that the resulting equation $v_0X_1v_1X_2 \dots X_n = X_1\omega_1X_2 \dots X_n\omega_n$, where v_i and ω_i are string constants, should hold for any $X_1, \dots, X_n$ values in the languages of the decisive fragments. We assume that the equation is minimal, that is, the equation cannot be split to lesser length-equal fragments. E.g. $aX_1bX_2 = X_1aX_2b$ can be definitely represented as a system of minimal equations $\begin{cases} aX_1 = X_1a \\ bX_2 = X_2b \end{cases}$, hence, it is not minimal.

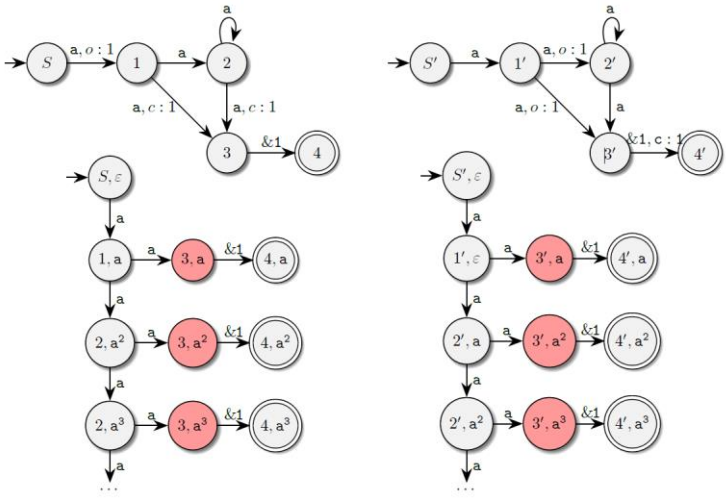


Fig. 5. Bisimilar MFA with non-bisimilar capture groups.

Equations of the form $v_0X_1v_1X_2 \dots X_n = X_1\omega_1X_2 \dots X_n\omega_n$ are well-studied [18]: any their solution X_i-component belongs to a simple regular language following the pattern $(\tau_1\tau_2)^n\tau_1$, i.e., the language of fractional powers of word $\tau_1\tau_2$. Therefore, bisimulation of MFAs with non-bisimilar capture groups including loops can occur, only if the loops themselves, in any their composition, are marked with powers of the same string.

In order to process bisimulation cases of the sort described above, we propose to use memory revision algorithm.

- All the decisive alternation fragments in capture groups are replaced with fresh string parameters.
- All the loops with no decisive alternations inside them (i.e., iterating along the constant path ξ) are replaced with the parameterized word ξ^{k_i}, where k_i is a fresh integer parameter or, in case of nested loops along the same subwords, parameter expression.
- The resulting values are unified, i.e., the parameterized and constant values are substituted instead of the path fragments. If the result of the substitution is a trivial equality, then the bisimulation can hold, otherwise, there is at least one path along action-bisimilar states that results in different values of $\&1_{A_1}$ and $\&1_{A_2}$.

Regarding nested loops, the procedure depends on their semantics. If given a loop with a minimal path reading $\xi_1\xi_2$, this loop contains an inner loop starting in ξ_1-th position, the inner loop contains a decisive alternation unless it iterates along the constant path $\xi_2\xi_1$ or a power of its primitive root ξ; in the latter case, words read along the both loops (the inner and the outer one) can be mapped to

a single parameterized word $\xi^{k_{i_1}+k_{i_2}}$, where k_{i_1} is a parameter denoting the outer iterations count, and k_{i_2} denotes the total inner iterations count.

Only if all the decisive actions are synchronized wrt the order induced by the synchronization, all the constant memory annotations are synchronized wrt each other, and all the memories are revised to be equal before referencing to them, then we can state that $A_1 \sim A_2$.

A prototype of the described algorithm, as well as the ref-word — MFA conversion and a fuzz equivalence testing module for MFA, is developed in the Chipollino formal language converter https://github.com/OnionGrief/Chipollino — the application that allows you to generate, transform and analyze various representations of formal languages (e.g., regular expressions, automata: FA, MFA, PDA). The MFA can be input by hand via a simple DSL language, or constructed from ref-words using Schmid algorithm [14], and an analogue of the Glushkov construction, merging ϵ-closures; random MFA and ref-words generators are supported in the project as well.

5. Multiple Memory Cells

In the case of multiple memory cells, the bisimulation problem meets new challenges, since the corresponding MFA paths can include references inside capture groups, as well as iterations over re-captured references. In the case of a single memory cell, at least we can assume that the capture groups used by bisimilar reference actions are action-equivalent (i.e., the languages of the corresponding action NFAs coincide). When the reference actions can occur inside the capture groups, that statement is not true. For example, values of capture groups with no decisive actions can behave the same way as the constant strings. An example of ref-words producing MFA with non-equivalent traces inside the capture groups is the pair $a[_1 a]_1 a[_2 \&1 a]_2 \&2$ and $[_1 a]_1 a[_2 a \&1]_2 a \&2$. The value stored in the cell 1 is a fixed constant, and its impact on the value stored in the cell 2 is exactly the same as in the case when this constant is explicitly stored.

If the memory cells have a cyclic dependency [19], the bisimulation problem very likely becomes undecidable, because the languages of the memory cells have even more expressible power than languages of the whole ref-words. An example is the ref-word $\left([_2 a \&1 b]_2 [_1 a \&2 b]_1\right)^*$: its memory cells 1 and 2 store the languages $\{a^{2n} b^{2n} | n \in \mathbb{N}\}$ and $\{a^{2n+1} b^{2n+1} | n \in \mathbb{N}\}$ respectively, and both can be proved to be inexpressible by any ref-word. Actually, given any linear context-free language L, a memory cell language can capture L; and, provided sequences $[_1 \Phi]_1 \Psi \&1, \Phi[_1 \Psi]_1 \&1$ depending on words in generated by these languages, the bisimulation between them holds if and only if the word equation $\Phi = \Psi$ for members of these languages is always true. This observation shows that the multi-cell bisimulation is definitely a hard problem, and maybe even undecidable.

6. Conclusion

The bisimulation problem of memory finite automata appears to be tractable at least in some practical cases. If a bisimulation is constructed on the states of A itself, then the bisimilar nodes in A can be merged with no change of the captured values, or the MFA traces. While the minimization problem for MFA is undecidable, the optimisation by bisimulation can be a decent approximation of the minimization, especially in the case when the MFA is deterministic. Efficiency estimation of this optimization is a future work of our MFA project.

Another interesting question is the decidability and complexity issue of MFA bisimulation in the general case. The existential theory of strings is known to be decidable [20-21], however, the bisimulation problem requires an algorithm not to decide a sole question whether a word equation has at least one solution, but to check if the language generated by the previous traces of MFA always satisfies this equation.

References

[1]. I. Free Software Foundation. (1993–2024) The GNU ed line editor. [Online]. Available: https://www.gnu.org/software/ed/manual/edmanual.html

[2]. D. D. Freydenberger, “Inclusion of pattern languages and related problems,” Ph.D. dissertation, Goethe University Frankfurt am Main, 2011. [Online]. Available: http://publikationen.ub.uni-frankfurt.de/frontdoor/index/index/docId/22351

[3]. T. Jiang, A. Salomaa, K. Salomaa, and S. Yu, “Inclusion is undecidable for pattern languages,” in Automata, Languages and Programming, A. Lingas, R. Karlsson, and S. Carlsson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 301–312.

[4]. Google. (2010–2023) Official public repository of RE2 library. [Online]. Available: https://github.com/google/re2

[5]. C. Bianchini, B.Riccardi, A. Policriti, and R.Romanello, “Incremental NFA minimization,” in CEUR Workshop Proceedings, 2022.

[6]. C. Fu, Y. Deng, D. N. Jansen, and L. Zhang, “On equivalence checking of nondeterministic finite automata,” in Dependable Software Engineering. Theories, Tools, and Applications, K. G. Larsen, O. Sokolsky, and J. Wang, Eds. Cham: Springer International Publishing, 2017, pp. 216–231.

[7]. F. Bonchi and D. Pous, “Checking NFA equivalence with bisimulations up to congruence,” SIGPLAN Not., vol. 48, no. 1, p. 457–468, jan 2013. [Online]. Available: https://doi.org/10.1145/2480359.2429124

[8]. C. Stirling, “Decidability of bisimulation equivalence for normed pushdown processes,” Theoretical Computer Science, vol. 195, no. 2, pp. 113–131, 1998, concurrency Theory. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304397597002168

[9]. M. Benedikt, S. Goller, S. Kiefer, and A. S. Murawski, “Bisimilarity of pushdown automata is nonelementary,” in 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science, 2013, pp. 488–498.

[10]. L. D’Antoni and M. Veanes, “Forward bisimulations for nondeterministic symbolic finite automata,” in Tools and Algorithms for the Construction and Analysis of Systems, A. Legay and T. Margaria, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 518–534.

[11]. M. Berglund and B. van der Merwe, “Re-examining regular expressions with backreferences,” Theoretical Computer Science, vol. 940, pp. 66–80, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304397522006570

[12]. J. L. Peterson, Petri Net Theory and the Modelling of Systems. Prentice-Hall, April 1981.

[13]. L. Aceto, A. Ingolfsdottir, and J. Srba, The algorithmics of bisimilarity, ser. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2011, pp. 100–172. [Online]. Available: https://doi.org/10.1017/CBO9780511792588.004

[14]. M. L. Schmid, “Characterising REGEX languages by regular languages equipped with factor-referencing,” Information and Computation, vol. 249, pp. 1–17, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0890540116000109

[15]. P. Hazel. (1997–2021) PCRE2 electronic manual. [Online]. Available: https://www.pcre.org/current/doc/html/index.html

[16]. D. D. Freydenberger and M. L. Schmid, “Deterministic regular expressions with back-references,” Journal of Computer and System Sciences, vol. 105, pp. 1–39, 2019. [Online]. Available: https://doi.org/10.1016/j.jcss.2019.04.001

[17]. V. M. Glushkov, “The abstract theory of automata,” Uspekhi Mat. Nauk, vol. 16, pp. 3–62, 1961. [Online]. Available: http://mi.mathnet.ru/rm6668

[18]. J. D. Day, F. Manea, and D. Nowotka, “The hardness of solving simple word equations”, 42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017). Leibniz International Proceedings in Informatics (LIPIcs), Volume 83, pp. 18:1-18:14, Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2017) Available: https://doi.org/10.4230/LIPIcs.MFCS.2017.18

[19]. D. Ismagilova and A. Nepeivoda, “Disambiguation of regular expressions with backreferences via term rewriting,” Modeling and Analysis of Information Systems, vol. 31 iss. 4, pp. 426-445, 2024. [Online]. https://doi.org/10.18255/1818-1015-2024-4-426-445

[20]. J. D. Day, V. Ganesh, and F. Manea, “Formal languages via theories over strings: An overview of some recent results,” Bull. EATCS, vol. 140, 2023. [Online]. Available: http://eatcs.org/beatcs/index.php/beatcs/article/view/765

[21]. G. S. Makanin, “The problem of solvability of equations in a free semigroup,” Mat. Sb. (N.S.), vol. 103(145), pp. 147–236, 1977.

Информация об авторах / Information about authors

Антонина Николаевна НЕПЕЙВОДА – научный сотрудник Института программных систем РАН. Сфера научных интересов: теория формальных языков, программная семантика, математическая логика и функциональное программирование.

Antonina Nikolaevna NEPEIVODA – researcher in the Program Systems Institute of RAS. Research interests: formal language theory, program semantics, mathematical logic, and functional programming.

Александр Дмитриевич ДЕЛЬМАН – студент Московского государственного технического университета им. Н. Э. Баумана. Сфера научных интересов: конструирование компиляторов и обработка естественного языка.

Aleksandr Dmitrievich DELMAN – student of the Bauman Moscow State Technical University. Research interests: compiler design and natural language processing.

Анна Сергеевна ТЕРЕНТЬЕВА – студент Московского государственного технического университета им. Н. Э. Баумана. Сфера научных интересов: конструирование компиляторов.

Anna Sergeevna TERENTYEVA – student of the Bauman Moscow State Technical University. Research interests: compiler design and optimization.