



Bisimulations in Memory Finite Automata

Antonina Nepeivoda 

Aylamazyan Program Systems Institute of RAS
Pereslavl–Zalesskiy
a_nevod@mail.ru

Aleksandr Delman 

Bauman Moscow State Technical University
Moscow
adelman2112@gmail.com

Anna Terentyeva 

Bauman Moscow State Technical University
Moscow
mathhyn@gmail.com

Abstract—This work-in-progress 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 [1], [2], and encapsulate the expressiveness of the named capture groups. We propose an experimental algorithm for checking bisimulation of one-memory MFAs, and an algorithm sketch for checking MFA bisimulation with non-cyclic memories. For the latter, 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.

Index Terms—Memory finite automata, bisimulation, word equations, backreference

I. INTRODUCTION

Extended regular expressions have been known at least since the early 90s, when they were implemented in the text editor `ed` [3]. 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 recognised language. For example, the expression $(a^*)(b^*)\backslash1b\backslash2$ recognises the language $\{a^n b^m a^n b^m\}$, 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 memoized values and no loops and alternations in the core regex [4]. 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 [5] process even academic regular expressions via non-deterministic finite automata optimisations, because they can be much faster than the exact minimization algorithm, and can preserve the structure of the regular expressions. One of such NFA optimisation algorithms is merging the bisimilar state classes [6], [7], which is also a well-known technique in program optimisation. 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 [8]: 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) [9], [10]. 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 [11].

It seems very natural to consider bisimulation-based optimisations in the presence of capture groups, both because the bisimilarity is typically easier to compute than the language equivalence, and because the bisimulation-merging optimisations 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 optimisations 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 [12].

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 $\{a^n b^n\}$, which is both context-free and Petri net language, cannot be recognized by any backref-regex¹. On the other hand, the language $\{\omega\omega \mid \omega \in \{a, b\}^*\}$ is trivially captured by the regex $((a \mid b)^*)\backslash1$, while this language is known to be neither context-free nor Petri-net [13]. Thus, the backref-regexes formalism is independent from the classification of process algebras given in the paper [14].

¹While this language can be recognized by a capture group of a (recursive) backref-regex.

This work-in-progress 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 [1], [2], and encapsulate the expressiveness of the named capture groups with re-initializations. We propose an experimental algorithm for checking bisimulation of one-memory MFAs, and an algorithm sketch for checking MFA bisimulation for MFA with non-cyclic memories. For the latter, we show that, in some borderline cases, the bisimulation problem is closely related to a question whether a parameterized word is always a solution to a given word equation of an arbitrary form.

II. PRELIMINARIES

A. 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 II.1. Given labelled transition systems \mathcal{T}_1 , \mathcal{T}_2 and the action alphabet \mathcal{A} , bisimulation is a coarsest relation \sim between states of the systems satisfying the following property.

- if q_1 is a state in \mathcal{T}_1 , and q_2 is a state in \mathcal{T}_2 , and $\gamma \in \mathcal{A}$, and $q_1 \sim q_2$, then for every transition $q_1 \xrightarrow{\gamma} q'_1$ in \mathcal{T}_1 there is a transition $q_2 \xrightarrow{\gamma} q'_2$ in \mathcal{T}_2 s.t. $q'_1 \sim q'_2$, and vice versa.

The systems \mathcal{T}_1 and \mathcal{T}_2 are bisimilar iff their starting states are in bisimulation, and, in the case of the existence of final states, any final state in \mathcal{T}_1 is bisimilar to a final state in \mathcal{T}_2 , and vice versa.

State machines \mathcal{A}_1 and \mathcal{A}_2 are bisimilar iff 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 recognising the same language $\{a^{n+1} \mid n \in \mathbb{N}\}$ are given in Fig. 1.

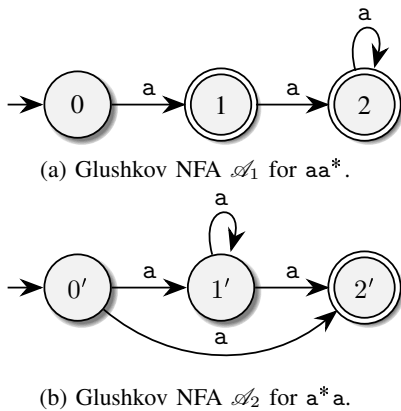


Fig. 1: Non-bisimilar equivalent NFA.

A simple and natural technique for checking the bisimilarity of the transition graphs \mathcal{T}_1 and \mathcal{T}_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 \mathcal{T}_1 and \mathcal{T}_2 respectively.
- Given the pair $\langle q_{k_i}, q_{k_j} \rangle$ Attacker chooses any element of the pair and a transition $q_r \xrightarrow{\gamma} q_p$ in the corresponding LTS. Defender must respond with a transition $q'_r \xrightarrow{\gamma} q'_p$ from the remaining state in the pair labelled with γ as well, and respecting the state q_p finality. Then the new player configuration consists of q_p and q'_p . Attacker can play again, if there are any transitions from q_p or q'_p .
- 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, \mathcal{A}_1) and play $0 \xrightarrow{a} 1$. Then, in order to respect the state finality, Defender is forced to reply with $0' \xrightarrow{a} 2'$ in \mathcal{A}_2 . Since $2'$ has no outgoing transitions, any action of Attacker in the LTS makes Defender to lose.

B. 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 [12]. In 2014, Markus Schmid suggested to consider only the 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 [12], 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 [1], [16], we also call extended regular expressions ref-words.

Definition II.2. Given an input alphabet Σ and the memory set cardinality $k \in \mathbb{N}$, a regular expression with backreferences (ref-word) is defined recursively:

- $\gamma \in \Sigma, \epsilon$, and $\&xi$, where $i \leq k$, are ref-words (the latter defines reading the i -th memory cell);
- if ρ_1 and ρ_2 are ref-words, then so are $(\rho_1 \mid \rho_2)$, $(\rho_1 \rho_2)$, $(\rho_i)^*$;
- if $i \leq k$ and ρ is a ref-word containing neither $\&xi$ nor $[i\tau]_i$, then $[i\rho]_i$ 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 [1], which admits unbalanced capture groups.

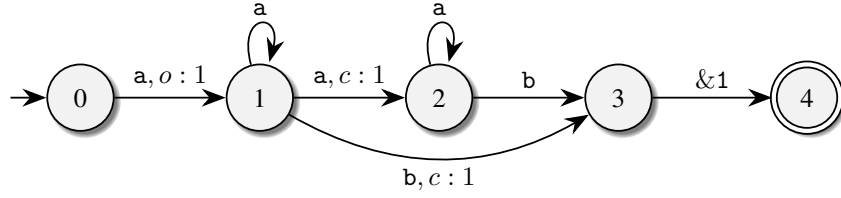


Fig. 2: MFA example for $[1aa^*]_1a^*b&1$.

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 [12], we say that we assume ε -semantics: all uninitialized references are valued ε . Thus, the ref-word given above recognises the language $\{ab^nab^n\}$.

C. Memory Finite Automata

A memory finite automaton (MFA) [1] is a tuple $\langle \mathcal{Q}, \Sigma, \delta, q_0, F \rangle$, where \mathcal{Q} is a finite set of states, Σ is the input alphabet, $q_0 \in \mathcal{Q}$ is a starting state, $F \subseteq \mathcal{Q}$ are final states, and $\delta : \mathcal{Q} \times (\Sigma \cup \{\varepsilon\} \cup \{1, 2, \dots, k\}) \rightarrow \mathcal{P}(\mathcal{Q} \times \{o, c, r, \diamond\}^k)$ is a transition table. The symbols o, c, r, \diamond are memory instructions (o — opening, c — closing, r — reset, \diamond — preserving instruction).

An MFA configuration is a tuple $(q, w, (u_1, r_1), \dots, (u_k, r_k))$, where q is a current state, w is an input to read, and for all $i, 1 \leq i \leq k$, (u_i, r_i) is an i -th memory state (u_i is a stored string; $r_i \in \{O, C\}$ is a memory status, which is either O or C). The initial memory state is $(q_0, w, (\varepsilon, C), \dots, (\varepsilon, C))$.

A transition from configuration $(q, vw, (u_1, r_1), \dots, (u_k, r_k))$ to $(p, w, (u'_1, r'_1), \dots, (u'_k, r'_k))$ is possible if there is a transition rule $(p, s_1, \dots, s_k) \in \delta(q, b)$, where either:

- $b \in \Sigma \cup \{\varepsilon\}$ and $v = b$,
- or $b \in \{1, 2, \dots, k\}$, and $s_b = c \vee r_b = C$ & $s_b = \diamond$ and $v = u_b$,

and for all i memory states change as follows:

- $(s_i = \diamond) \& (r_i = O) \Rightarrow (u'_i, r'_i) = (u_i v, r_i)$,
- $(s_i = \diamond) \& (r_i = C) \Rightarrow (u'_i, r'_i) = (u_i, r_i)$,
- $s_i = o \Rightarrow (u'_i, r'_i) = (v, O)$,
- $s_i = c \Rightarrow (u'_i, r'_i) = (u_i, C)$,
- $s_i = r \Rightarrow (u'_i, r'_i) = (\varepsilon, C)$.

An example of a memory finite automaton for a non-regular language $\{a^{n+k}ba^n \mid n > 0\}$ is given in Figure 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 preserving 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.

III. BISIMULATION IN MEMORY FINITE AUTOMATA

Definition III.1. Let $\mathcal{A} = \langle \mathcal{Q}, \Sigma, \delta, q_0, F \rangle$ be a memory finite automaton over k memory cells. Its transition graph $\mathcal{G}(\mathcal{A})$ is defined as follows.

- $(q_0, \underbrace{\langle (\varepsilon, C), \dots, (\varepsilon, C) \rangle}_k)$ is the starting node of the transition graph;
- given a configuration $N_i = (q, \langle (u_1, r_1), \dots, (u_k, r_k) \rangle)$, children of 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 \mathcal{A} are determined by the states (i.e., given a state q , the values of r_1, \dots, r_k can be restored independently of the previous trace). We assume that the condition holds for all MFA considered, and omit r_i values in the node configurations of transition graphs. While the notion “capture groups” is usually used in the context of ref-words, given an MFA \mathcal{A} , we also say that the subgraphs between the open and close operations wrt the cell k are “capture groups” for the reference k in \mathcal{A} . Moreover, 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 in distinct \mathcal{A}_1 and \mathcal{A}_2 with the same label, we sometimes mark the references read in the transition graphs with the corresponding subscripts, i.e. $&k_{\mathcal{A}_1}$ and $&1_{\mathcal{A}_2}$.

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

Assumption III.1. Given the transition graphs $\mathcal{G}(\mathcal{A}_1)$ and $\mathcal{G}(\mathcal{A}_2)$, the transitions from nodes N_{j_1} and N_{j_2} both labelled with $&i$ are equal iff u_i value in the configuration of N_{j_1} coincides with 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 III.2. Given MFA \mathcal{A}_1 and \mathcal{A}_2 , the MFA are bisimilar (denoted $\mathcal{A}_1 \sim \mathcal{A}_2$), iff their transition graphs $\mathcal{G}(\mathcal{A}_1)$ and $\mathcal{G}(\mathcal{A}_2)$ 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

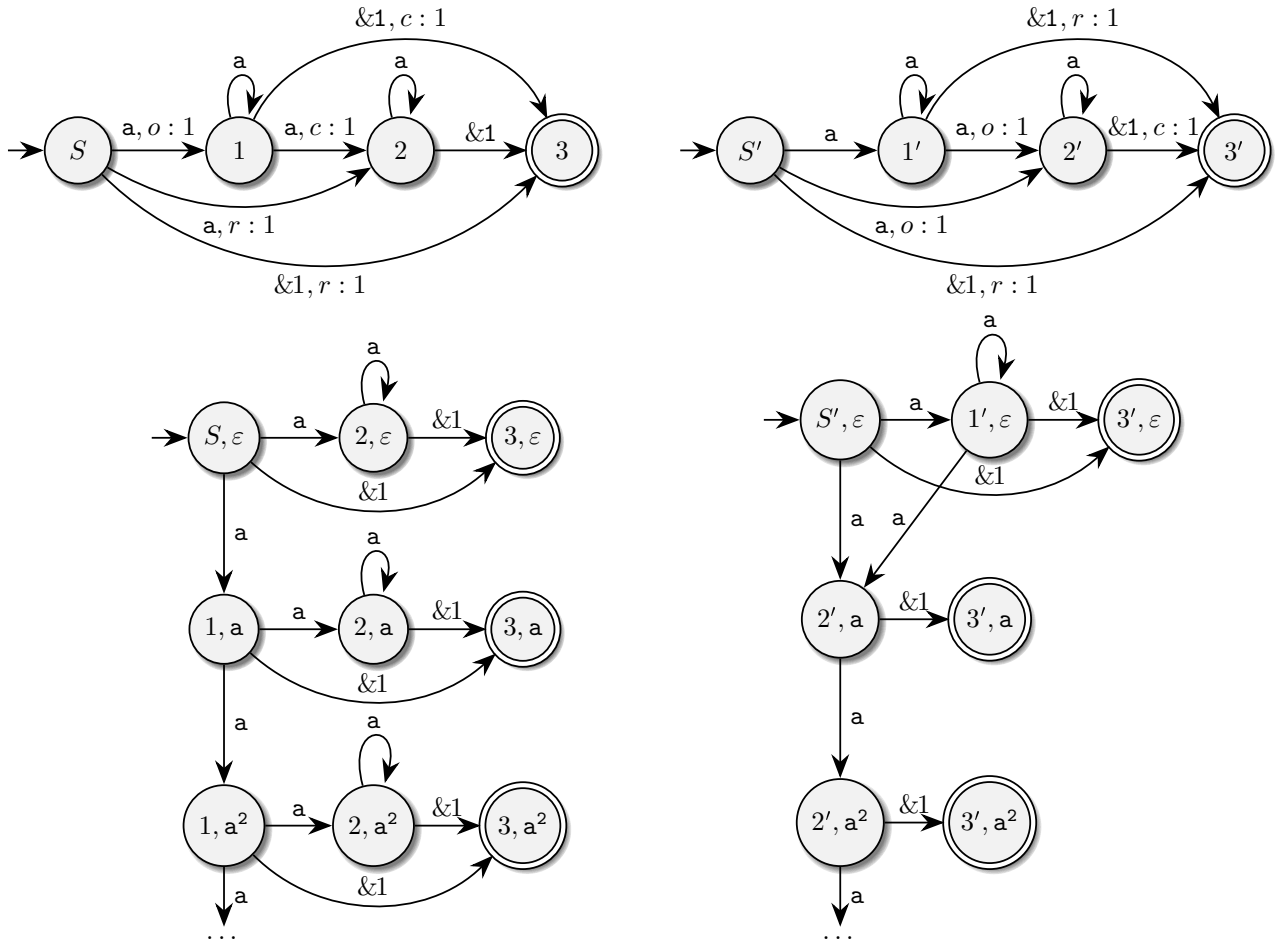


Fig. 3: Non-bisimilar MFA for $[1a^*]_1a^*\&1$ and $a^*[1a^*]_1\&1$ and their transition graphs.

an action automaton, which describes possible sequences of control actions in the process graph of a MFA.

Definition III.3. Given k -cell MFA \mathcal{A} , its action NFA $\pi_{\mathcal{M}}(\mathcal{A})$ results from \mathcal{A} by erasing memory operations from edge labels of \mathcal{A} . The symbols $\&i$ ($1 \leq i \leq k$) become elements of the input action alphabet \mathcal{A} for $\pi_{\mathcal{M}}(\mathcal{A})$.

If MFA \mathcal{A}_1 and \mathcal{A}_2 are in the bisimulation relation, then the control traces of them must coincide, thus, $\pi_{\mathcal{M}}(\mathcal{A}_1) \sim \pi_{\mathcal{M}}(\mathcal{A}_2)$. In the latter case, we say that \mathcal{A}_1 and \mathcal{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 $\&k_{\mathcal{A}_1}$ played by Attacker, not only Defender must respond with the action $\&k_{\mathcal{A}_2}$, 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 (and even equal), but Attacker has the following winning strategy.

- Play $S' \xrightarrow{a} 1'$.
- If Defender responds with $S \xrightarrow{a} 1$, then play $1' \xrightarrow{\&1} 3'$ and

make Defender to lose, since they cannot 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.

IV. BISIMULATION FOR ONE-MEMORY-CELL 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 an alternation by different $\gamma_1, \gamma_2 \in \Sigma$ (maybe, preceded with a common prefix action ω) inside a capture group, decide which branch to choose.

Henceforth we call the two sorts of actions the decisive actions, by default assuming that the actions occur inside the capture group. If the actions are considered outside the capture groups, we mention this fact explicitly.

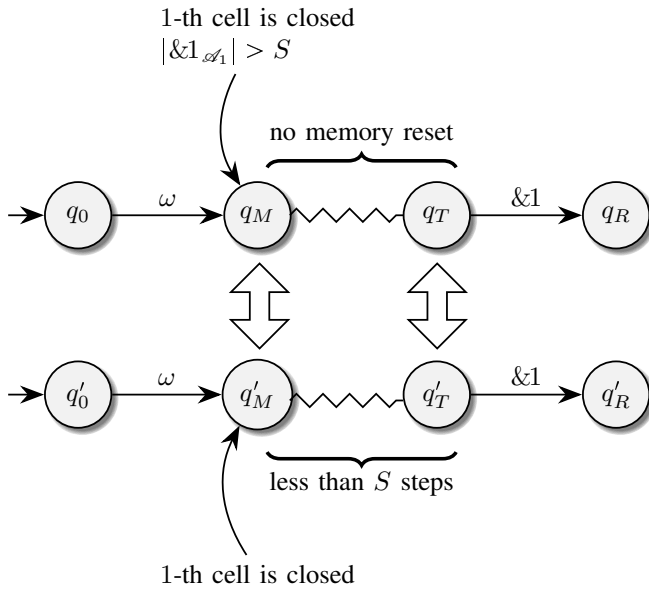


Fig. 4: Strategy for Attacker in case of a decisive loop action.

Proposition IV.1. *Let $\pi_{\mathcal{M}}(\mathcal{A}_1) \sim \pi_{\mathcal{M}}(\mathcal{A}_2)$. If there exists a state q_1 in \mathcal{A}_1 s.t. it has an outgoing decisive action (inside a capture group) and none of the states q_i in \mathcal{A}_2 that are action-bisimilar to q_1 has such a decisive action inside the capture group, then $\neg(\mathcal{A}_1 \sim \mathcal{A}_2)$.*

Proof. Let us show a winning strategy for Attacker under the proposition conditions in the case of the decisive alternation. Let $\&l$ in $\mathcal{G}(\mathcal{A}_2)$ be initialized with a value with no γ_1 in k -th position, and $\&l$ in $\mathcal{G}(\mathcal{A}_1)$ be initialized with ξ . Choose the path along γ_1 , thus, accumulating the value $\xi\omega\gamma_1$ in $\&l$. If there is at least one path in \mathcal{A}_1 leading to action $\&l$ s.t. it is not action-bisimilar to a path with a re-initialization of $\&l$ in \mathcal{A}_2 , move along the path and make Defender to lose, not being able to reset the memory cell. Otherwise, due to the usefulness of the capture groups, there is at least one path that has no reset of $\&l$ in \mathcal{A}_1 , and has a state, that is action-bisimilar to a state with a memoized outgoing decisive alternation between γ_1 and γ_2 in \mathcal{A}_2 , which occurs on the trace with the previously accumulated memory value ξ . Given the alternation, Attacker chooses the path containing γ_2 , thus, forcing of memoization of the prefix $\xi\omega\gamma_2$ in the memory cell along the trace \mathcal{A}_2 .

Now let us show Attacker's winning strategy in the case of the decisive loop action in \mathcal{A}_1 which is not action-bisimilar to any decisive action in \mathcal{A}_2 . First, Attacker decides to play the looping back action at least S times, where S is the number of states in \mathcal{A}_1 , and additionally chooses such a number of iterations that the value of $|\&l|_{\mathcal{A}_1}$ is not equal to $\&l_{\mathcal{A}_2}$. Second, Attacker chooses the shortest possible path in \mathcal{A}_1 to the action $\&l$. In order to imitate the action $\&l_{\mathcal{A}_1}$, Defender is forced to reset the value of $\&l_{\mathcal{A}_2}$, but the value of $\&l_{\mathcal{A}_1}$ is longer than any possible value read along at most S 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. \square

Proposition IV.1 states that Defender has a chance to find a winning strategy, only if all decisive actions in action-bisimilar states of \mathcal{A}_1 and \mathcal{A}_2 occur synchronously. Moreover, in Figure 3, while the states 1 and 2' containing decisive actions are action-bisimilar, the process graphs are not. The reason is that Proposition IV.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 \mathcal{A}_1 , allowing Attacker to read additional letters a after the memory is closed, while in \mathcal{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.

It is tempting to conclude that the whole capture groups of the MFA are required to be action-bisimilar in order to guarantee that $\mathcal{A}_1 \sim \mathcal{A}_2$. Nevertheless, non-decisive actions can occur asynchronously without losing the bisimulation.

An example of such a bisimulation 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 $\&l_{\mathcal{A}_1}$ and $\&l_{\mathcal{A}_2}$ always coincide in the states 3 and 3'. In order to process such bisimulation cases, 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 $\&l_{\mathcal{A}_1}$ and $\&l_{\mathcal{A}_2}$, and $\neg(\mathcal{A}_1 \sim \mathcal{A}_2)$.

Regarding nested loops, the procedure depends on their semantics. If given a loop reading $\xi_1\xi_2$, the loop contains an inner loop starting in $|\xi_1|$ -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; in the latter case, both loops (the inner and the outer one) can be mapped to a single parameterized word $\xi^{f(k_{i_1}, k_{i_2})}$, where k_{i_1} is a parameter

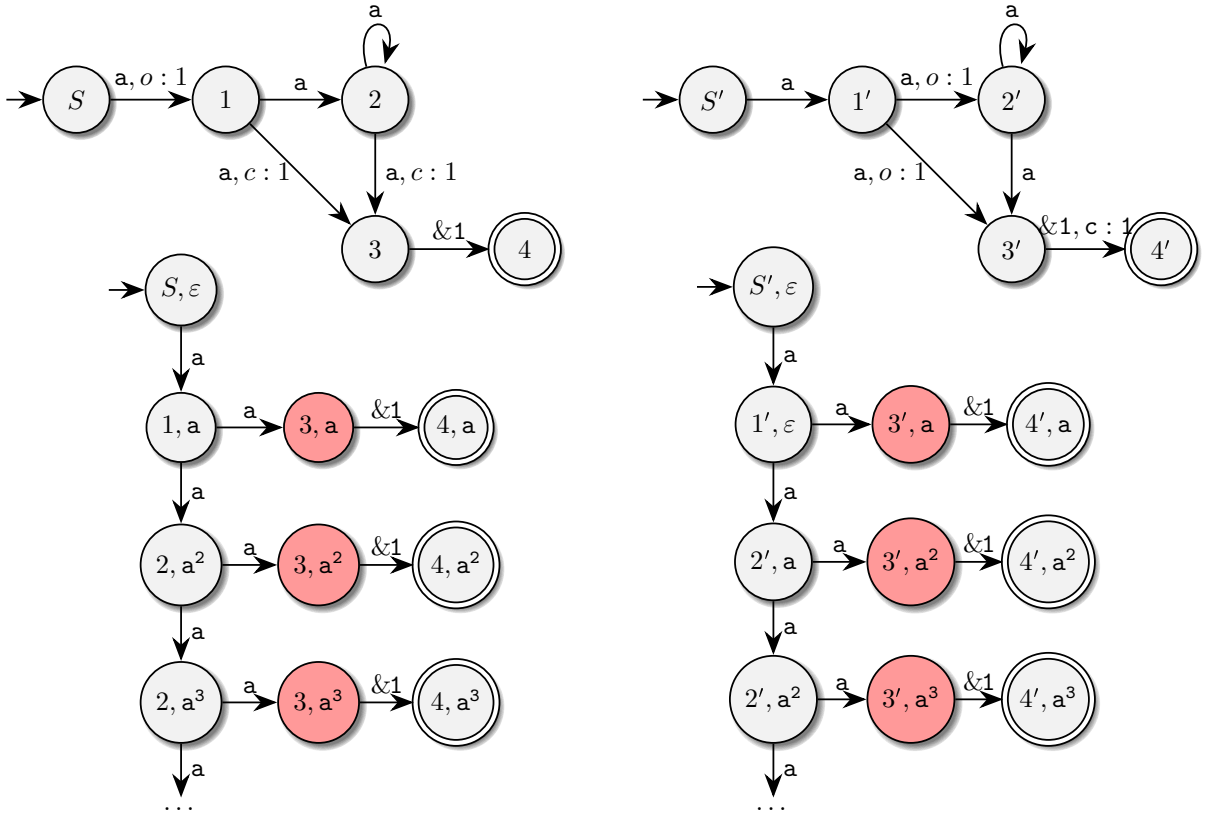


Fig. 5: Bisimilar MFA for $[1aa^*]_1a&1$ and $a[1a^*a]_1&1$ with non-bisimilar capture group 1.

denoting the outer iterations count, and k_{i_2} denotes the inner iterations count.

If and only if all the decisive actions are synchronised wrt the order induced by the synchronisation, and all the memories are revised to be equal before referencing to them, then we can state that $\mathcal{A}_1 \sim \mathcal{A}_2$.

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 [1], and an analogue of the Glushkov construction, merging ε -closures; random MFA and ref-words generators are supported in the project as well.

V. MULTIPLE MEMORY CELLS

In the case of multiple memory cells, the bisimulation problem meets new challenges, since references inside capture groups, as well as iterations over references, become possible. 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 occur

inside the capture groups, that statement is not true. The capture groups in the bisimilar MFA must memoize equal sequences of decisions, but the decisions can be combined in multiple ways, still resulting in the same reference values. Moreover, memory values of capture groups with no decisive actions can behave the same way as the constant strings. In Fig. 6, an example of bisimilar MFA with non-equivalent traces inside the capture groups is given, because the value of $&1$ is fixed, and its impact on the memory value $&2$ is indistinguishable from the impact when reading a constant letter.

If the memory cells are acyclic (i.e. do not depend on each other values recursively²), then a prototype algorithm for checking MFA bisimilarity can still be suggested.

- Construct state bisimulation of $\pi_{\mathcal{M}}(\mathcal{A}_1)$ and $\pi_{\mathcal{M}}(\mathcal{A}_2)$.
- Introduce the decision dependency relation: a memory cell $&k_i$ preserves a decision done in $&k_j$, if the decisive action in the capture group $&k_j$ is also captured by k_i -th memory, and the decisive actions are equally reachable from the decisive actions previously occurring in the automata.
- Colour the decisive actions in \mathcal{A}_1 and \mathcal{A}_2 and compute the decision dependencies relation closure wrt each decision. If the closures for the capture groups $&k_{\mathcal{A}_1}$ and

²The notion of the acyclic memory dependency is discussed in the paper [18] and is also implemented in the converter.

$&k_{\mathcal{A}_2}$ coincide, then perform the memory revisions on the capture groups. Now the parameterized words can be finitely iterated.

- Bisimulation holds iff all the memories are revised to be equal before referencing.

However, if the memory cells have a cyclic dependency, then the algorithm above cannot be applied, because some resulting memory values could not be represented with parameterized words [19], [20].

For example, given the following ref-words,

$$\begin{aligned} &[_1b^*]_1([_2(\&1ba)^*\&1b]_2[_1(\&2ab)^*\&2a]_1)^* \\ &\quad [_3\&2ab\&1]_3\&1ba\&2\&3 \\ &[_1b^*]_1([_2(\&1ba)^*\&1b]_2[_1(\&2ab)^*\&2a]_1)^* \\ &\quad \&2ab\&1[_3\&1ba\&2]_3\&3 \end{aligned}$$

their Glushkov MFA are bisimilar, because any two words ω_1 valuing $\&1$ and ω_2 valuing $\&2$ satisfy the equation $\omega_1 a b \omega_2 = \omega_2 b a \omega_1$. Since any word equation can be modelled with the bisimulation problem in the same sense, it is not clear how hard is the MFA bisimulation problem in the general case.

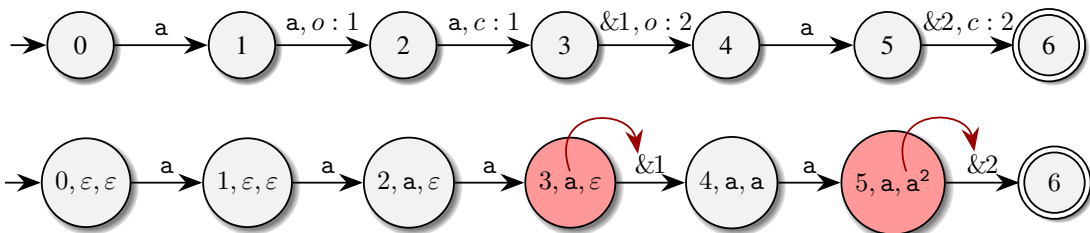
VI. CONCLUSION

The bisimulation problem of memory finite automata appears to be tractable at least in many practical cases, namely, if the memory statuses of the nodes are determined, and the memory dependencies are acyclic. If a bisimulation is constructed on the states of \mathcal{A} itself, then the bisimilar nodes in \mathcal{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 optimisation is a future work of our MFA project.

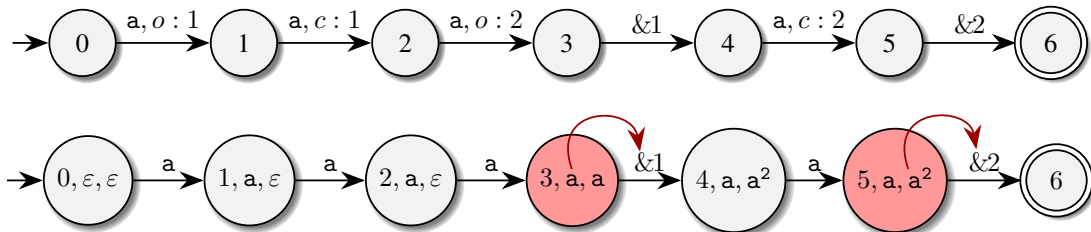
Another interesting question is the decidability and complexity issue of MFA bisimulation in the general case. The existential theory of strings is known to be decidable [20], [21], however, the bisimulation problem requires an algorithm not to decide a sole question whether a word equation has at least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

REFERENCES

- [1] 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>
- [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] I. Free Software Foundation. (1993–2024) The GNU ed line editor. [Online]. Available: https://www.gnu.org/software/ed/manual/ed_manual.html
- [4] 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.
- [5] Google. (2010–2023) Official public repository of RE2 library. [Online]. Available: <https://github.com/google/re2>
- [6] C. Bianchini, B. Riccardi, A. Policriti, and R. Romanello, "Incremental NFA minimization," in *CEUR Workshop Proceedings*, 2022.
- [7] 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.
- [8] 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>
- [9] 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>
- [10] M. Benedikt, S. Göller, 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.
- [11] 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.
- [12] 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>
- [13] J. L. Peterson, *Petri Net Theory and the Modelling of Systems*. Prentice-Hall, April 1981.
- [14] L. Aceto, A. Ingólfssdóttir, 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>
- [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] D. Ismagilova and A. Nepeivoda, "Disambiguation of regular expressions with backreferences via term rewriting," *MAIS*, to appear.
- [19] E. Czeizler, "The non-parametrizability of the word equation $xyz=zvx$: a short proof," *Theor. Comput. Sci.*, vol. 345, no. 2–3, pp. 296–303, nov 2005. [Online]. Available: <https://doi.org/10.1016/j.tcs.2005.07.012>
- [20] G. S. Makanin, "The problem of solvability of equations in a free semigroup," *Mat. Sb. (N.S.)*, vol. 103(145), pp. 147–236, 1977.
- [21] 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>



(a) MFA and its transition graph for $a[1a]1a[2\&1a]2\&2$.



(b) MFA and its transition graph for $[1a]1a[2a\&1]2a\&2$.

Fig. 6: Bisimulation of MFA with non-equivalent traces in capture groups.