Efficient Testing of Equivalence of Words in a Free Idempotent Semigroup*

Jakub Radoszewski[1] **and Wojciech Rytter**[1,2]

[1] Department of Mathematics, Computer Science and Mechanics,
University of Warsaw, Warsaw, Poland
[jrad,rytter]@mimuw.edu.pl
[2] Faculty of Mathematics and Informatics,
Copernicus University, Toruń, Poland

Abstract. We present an automata–theoretic approach to a simple Burnside–type problem for semigroups. For two words of total length n over an alphabet Σ, we give an algorithm with time complexity $O(n \cdot |\Sigma|)$ and space complexity $O(n)$ which tests their equivalence under the idempotency relation $x^2 \approx x$. The algorithm verifies whether one word can be transformed to another one by repetitively replacing any factor x^2 by x or z by z^2. We show that the problem can be reduced to equivalence of acyclic deterministic automata of size $O(n \cdot |\Sigma|)$. An interesting feature of our algorithm is small space complexity — equivalence of introduced automata is checked in space $O(n)$, which is significantly less than the sizes of the automata. This is achieved by processing the acyclic automata layer by layer, each layer only of size $O(n)$, hence only small part of a large virtual automaton is kept in the memory.

Key words: Burnside–type problem, finite automata, efficient algorithm.

1 Introduction

In this paper we study algorithmic aspects of some problems related to Burnside–type problems in semigroups. In 1902, Burnside [2] raised the following famous problem: "Is every group with a finite number of generators and satisfying an identical relation $x^r \approx 1$ finite?". Although the problem was solved negatively in 1968 by Adjan and Novikov [1], it has given birth to several related problems, including the Burnside problem for semigroups. The problem was first studied

* Supported by grant N206 004 32/0806 of the Polish Ministry of Science and Higher Education.

by Green and Rees, who proved [8] in 1952 that a finitely generated semigroup satisfying the identity $x^{r+1} \approx x$ is finite provided any finitely generated group satisfying the identity $x^r \approx 1$ is finite. In particular the free idempotent semigroup, i.e., satisfying the identity $x^2 \approx x$ is finite.

Although the theory of free Burnside semigroups was developed much slower than the corresponding theory for groups, tremendous progress was achieved in the former in the last 15 years — a summary of the known results can be found in the excellent survey by do Lago and Simon [7]. In general, the semigroups satisfying $x^{r+s} \approx x^r$ for $r, s \geq 1$ are analyzed from different points of view: finiteness, regularity of languages corresponding to congruence classes of $\approx$, the structure of maximal subgroup of the semigroup and finally the *word problem*, in which it is investigated whether testing of $u \approx v$ is decidable. For $r \geq 3$ the word problem was proved to be decidable (due to the work of several authors [3–6, 9, 10, 15]), for $r = 2$ the problem remains open (although for some cases effective algorithms were established, as in the recent paper [16]), finally for $r = 1$ the decidability of the word problem for groups implies decidability for semigroups [11].

To the best of our knowledge, the problem of efficient implementation of the algorithm for the word problem under idempotency relation ($r = 1$) has never been studied previously. For words $u, v \in \Sigma^*$ such that $|u|+|v| = n$, we design an algorithm with time complexity $O\big(n \cdot |\Sigma|\big)$ and memory complexity $O\big(n\big)$. Here we introduce an intuitive assumption that $|\Sigma| = O(n)$. This result is far better than a straightforward dynamic programming solution which yields $O\big(n^5\big)$ time complexity. Additionally, an interesting part of our result is that it combines methods from algebra, finite automata and algorithm analysis.

Let Σ be an arbitrary finite alphabet, $|\Sigma| = K$. Let Σ^* be the set of all words over the alphabet Σ and let $\Sigma^+ = \Sigma^* \setminus \{\epsilon\}$ where ϵ is the empty word. We introduce the *idempotency relation* $\sim_i$ in Σ^*

$$\forall_{x \in \Sigma^*} \quad xx \sim_i x$$

and denote by $\approx$ the congruence it generates. Then, the *free idempotent semigroup* (also called the free band) M generated by Σ is the set

$$M = \Sigma^* / \approx \ .$$

There also exists an alternative definition of M that is more relevant to our paper. We say that two words $u, v \in \Sigma^*$ are equivalent ($u \approx v$) if v can be derived from u, and vice versa, by a finite (possibly 0) number of applications of the rules:

– replace a factor x of u by its square xx, or
– replace a square factor xx by the word x.

Relation $\approx$ is an equivalence relation and the set of equivalence classes of this relation forms a semigroup (under concatenation) that is isomorphic to M. Due to the Green–Rees theorem, the set M generated by any finite set Σ is also finite. The proof of the theorem [8, 12–14] not only specifies its cardinality

$$\sum_{i=0}^{K} \binom{K}{i} \prod_{1 \le j \le i} (i-j+1)^{2^j}$$

but also provides a recursive criterion for verification of equivalence of u and v under $\approx$ (see Theorem 1).

2 An abstract algorithm and factor automata

For $u = u_1 \dots u_k$, by $u[i..j]$ we denote a factor of u equal to $u_i \dots u_j$ (in particular $u[i] = u[i..i]$) and by $|u|$ we denote length of u, i.e. k. Words $u[1..i]$ are called prefixes of u, and words $u[i..k]$ suffixes of u.

Let $\mathrm{Alph}(u)$ be the set of all letters appearing in u. With each $u \in \Sigma^+$ we associate a (characteristic) quadruple

$$u \stackrel{\circ}{=} (p, a, b, q), \quad \text{where:}$$

– $a, b \in \Sigma$, pa is a prefix and bq is a suffix of u;
– $\mathrm{Alph}(p) = \mathrm{Alph}(u) \setminus \{a\}$, and $\mathrm{Alph}(q) = \mathrm{Alph}(u) \setminus \{b\}$.

Example.

$$\texttt{ababbbcbcbc} \stackrel{\circ}{=} (\texttt{ababbb},\ \texttt{c},\ \texttt{a},\ \texttt{bbbcbcbc})$$

Theorem 1 (equivalence criterion). [8]
Assume $u \stackrel{\circ}{=} (p, a, b, q)$, $v \stackrel{\circ}{=} (p', a', b', q')$. *Then,*

$$u \approx v \quad \textit{iff} \quad (\ p \approx p' \ \wedge\ a = a' \ \wedge\ b = b' \ \wedge\ q \approx q'\ ).$$

The theorem implies correctness of the following abstract algorithm testing if $u \approx v$.

Algorithm TEST(u, v)
  if $\mathrm{Alph}(u) \neq \mathrm{Alph}(v)$ **then return** $false$
  if $\mathrm{Alph}(u) = \emptyset$ **then return** $true$
  let $u \stackrel{\circ}{=} (p, a, b, q)$, $v \stackrel{\circ}{=} (p', a', b', q')$
  if $a \neq a' \vee b \neq b'$ **then return** $false$
  return $\mathrm{TEST}(p, p') \wedge \mathrm{TEST}(q, q')$

Let $\overline{\Sigma}$ be a disjoint copy of the alphabet Σ. For each letter $a \in \Sigma$ denote by $\overline{a}$ its copy. For words u, v with the same set of letters we define the *factor automaton* $A_{u,v}$ as follows:

- The set of input symbols is $\Sigma \cup \overline{\Sigma}$ and the set of states is the set of all factors of u and v.
- If x is a factor and $x \triangleq (p, a, b, q)$ then we have two transitions

$$x \xrightarrow{a} p, \quad x \xrightarrow{\overline{b}} q \ .$$

  Other transitions are undefined.
- There is only one accepting state: the empty word ϵ.

Observation 1 *From every state of $A_{u,v}$ there exists an accepting path.*

Recall that two states x, y of an automaton are called *equivalent* (notation: $x \sim y$) if the sets of labels of all accepting paths starting at x and at y are equal.

Lemma 1. *$u \approx v$ iff u and v are equivalent as states of the factor automaton $A_{u,v}$.*

Proof. We prove that for any two states x, y, $x \sim y$ iff $x \approx y$, by induction on $\min(|x|, |y|)$ (here we refer to x and y both as states of $A_{u,v}$ and as words from Σ^*).

The basis is very simple: if $x = \epsilon$ or $y = \epsilon$ then $x \sim y$ iff both x and y are accepting states $x = y = \epsilon$, what is equivalent to $x \approx y$.

If $|x|, |y| > 0$ then let

$$x \triangleq (p, a, b, q), \quad y \triangleq (p', a', b', q') \ .$$

In x there are transitions

$$x \xrightarrow{a} p, \quad x \xrightarrow{\overline{b}} q$$

whereas in y:

$$y \xrightarrow{a'} p', \quad y \xrightarrow{\overline{b'}} q' \ .$$

Assume that $x \sim y$. Due to Observation 1, there exists an accepting path starting from p, consequently there exists an accepting path from x starting with a transition with label a. Because $a \neq \overline{b'}$, this implies that $a = a'$ and $p \sim p'$. Similarly, there exists an accepting path starting from q, so $\overline{b} = \overline{b'}$ and $q \sim q'$. Thus we proved that

$$x \sim y \quad \Rightarrow \quad a = a', \ b = b', \ p \sim p', \ q \sim q' \ .$$

Conversely, let us observe that none of the states x, y is accepting. Therefore, by definition

$$a = a',\ b = b',\ p \sim p',\ q \sim q' \quad \Rightarrow \quad x \sim y\ .$$

Combining both implications, we obtain that $x \sim y$ iff $a = a'$, $b = b'$, $p \sim p'$, $q \sim q'$. Due to the inductive hypothesis, the last two conditions are equivalent to $p \approx p'$, $q \approx q'$. We conclude the proof applying the criterion from Theorem 1. □

We reduced equivalence of factors to equivalence of states in a deterministic automaton, however this does not give directly an efficient algorithm since the definition of the automaton is rather abstract.

3 Interval automata: more efficient automata

We introduce interval automata as efficient implementation of factor automata. Because it is more convenient to deal with the same single word, we introduce the word $w = u\$v$, $|w| = n$, where \$ is a special delimiter that we add to Σ. From now on we deal only with the word w.

For a word w define the rank of an interval $[i..j]$ as $|\mathrm{Alph}(w[i..j])|$. We say that an interval $[i..j]$ is *k–left* (for a fixed j) iff i is the smallest number such that $[i..j]$ is of rank k, similarly we define the *k–right* interval $[i..j]$ (for a fixed i) as the one for which j is the largest number such that $[i..j]$ is of rank k.
The k–left and k–right intervals are called *k–intervals*, and their set is denoted by $\mathcal{I}_k$ and called here the *k^{th} layer*. Let $\mathcal{I}$ be the union of all $\mathcal{I}_k$'s.

We define *interval automaton* $G(w)$ as follows. The set of states is $\mathcal{I}$ and the input alphabet is the same as for the factor automaton. If

$$x \stackrel{\circ}{=} (p, a, b, q), \quad \text{where} \quad x = w[i..j], \quad p = w[i..k], \quad q = w[l..j]$$

then we have two transitions

$$[i..j] \xrightarrow{a} [i..k], \quad [i..j] \xrightarrow{\overline{b}} [l..j]\ .$$

Other transitions are undefined. There is only one accepting state: the empty interval $\emptyset$ corresponding to the empty word ϵ.

Lemma 2. *Intervals corresponding to u and v within $w = u\$v$ are states of the interval automaton $G(w)$ and are equivalent iff their corresponding states are equivalent in the factor automaton $A_{u,v}$.*

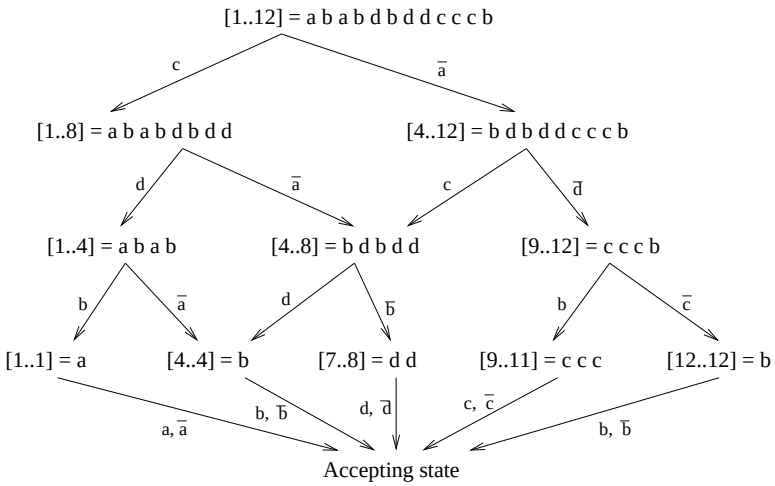

Fig. 1. The interval automaton of the word $w = \texttt{ababdbddcccb}$. For simplicity, the figure does not contain intervals from $\mathcal{I}$ that are not located on any path from $[1..12]$ to the accepting state $\emptyset$.

Proof. Let us note that intervals corresponding to u and v are $|\mathrm{Alph}(u)|$– and $|\mathrm{Alph}(v)|$–intervals in w, therefore they appear in $G(w)$. Moreover, if states representing factors containing the \$ symbol are omitted, $G(w)$ is a subautomaton of $A_{u,v}$ and contains only the states that are "important" w.r.t. the algorithm of testing if $u \approx v$. In particular, since the defined transitions in $G(w)$ always lead to intervals from $\mathcal{I}$, all factors of w accessible from u and v are present in $G(w)$. □

The k^{th} layer $\mathcal{I}_k$ will be represented by tables $LEFT_k,\ RIGHT_k$ where:

$LEFT_k[j] = i$ if $[i..j]$ is a k–left interval;

$RIGHT_k[i] = j$ if $[i..j]$ is a k–right interval;

if a corresponding k–left or k–right interval does not exist, $LEFT_k[j]$ and $RIGHT_k[i]$ are undefined.

Lemma 3.

a) For each k we can compute in time and space $O(n)$ the set of intervals of rank k represented by the tables $LEFT_k,\ RIGHT_k$.

b) The interval automaton $G(w)$ can be constructed in $O(n \cdot |\Sigma|)$ time layer by layer, starting from layer 0 and finishing in layer K, in such a way that

the construction of layer i requires only knowledge of layer $i-1$ and $O(n)$ additional storage.

Proof. To construct the table $RIGHT_k$ for a given $k \in \{0, 1, \ldots, K\}$ we use a sliding window algorithm (see the pseudocode of Compute_RIGHT1 and Fig. 2).

Algorithm Compute_RIGHT1(w, n, k)
```
j ← 0
Z ← ∅   (* a multiset *)
for i = 1, 2, ..., n do
    if i > 1 then Z ← Z \ {w[i − 1]}
    while j < n and |Z ∪ {w[j + 1]}| ≤ k do
        j ← j + 1
        Z ← Z ∪ {w[j]}
    if |Z| = k  then RIGHT_k[i] ← j
```

In the i^{th} step of the **for** loop of the algorithm we compute the *multiset* of characters

$$Z = \{w[i], w[i+1], \ldots, w[j]\} \quad \text{such that} \quad RIGHT_k[i] = j$$

using the observation that for each i, $RIGHT_k[i+1] \geq RIGHT_k[i]$.

If Z is implemented as a count array of size $K = O(n)$, it can be initialized in $O(n)$ time and all necessary operations on Z — inserting elements, deleting elements and computing the number $|Z|$ of *different* letters present in Z — can be performed in $O(1)$ time. The following pseudocode is an implementation of algorithm Compute_RIGHT1 using a count array.

Algorithm Compute_RIGHT2(w, n, k)
```
Z : array[1..K] = (0, 0, ..., 0)
size ← 0;  j ← 0
for i = 1, 2, ..., n do
    if i > 1 then   (* performing the assignment "Z ← Z \ {w[i − 1]}" *)
        Z[w[i − 1]] ← Z[w[i − 1]] − 1
        if Z[w[i − 1]] = 0  then size ← size − 1
    while j < n and (Z[w[j + 1]] ≠ 0 or size < k) do
        j ← j + 1
        if Z[w[j]] = 0  then size ← size + 1
        Z[w[j]] ← Z[w[j]] + 1
    if size = k  then RIGHT_k[i] ← j
```

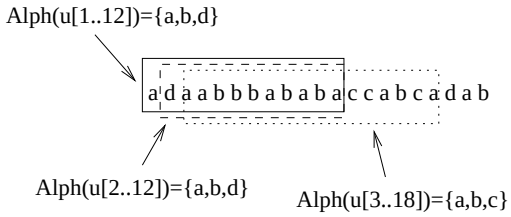

Fig. 2. Sliding window appearing during the computation of $RIGHT_3[1] = 12$, $RIGHT_3[2] = 12$ and $RIGHT_3[3] = 18$ for the word `adaabbbababaccabcadab`.

The total number of steps of the **while** loop of the algorithm is $O(n)$, since in each step j increases by one. Thus, the total time and memory complexity of Compute_RIGHT2 is $O(n)$.

Computation of $LEFT_k$ can be performed analogically, what concludes the proof of point a).

Point b) follows from a), since transitions from interval $[i..j]$ in the interval automaton lead to intervals

$$[i..RIGHT_k[i]] \quad \text{and} \quad [LEFT_k[j]..j]$$

where $k + 1 = |\text{Alph}(w[i..j])|$, and are labeled with letters

$$w[RIGHT_k[i] + 1] \quad \text{and} \quad \overline{w[LEFT_k[j] - 1]}$$

respectively. □

4 Testing equivalence of states in $G(w)$

Our final goal is to design an algorithm for testing equivalence of states of the interval automaton $G(w)$. Let us first note that $G(w)$ also has the property mentioned in Observation 1.

Observation 2 *From every state of $G(w)$ there exists an accepting path.*

Lemma 4. *If x, y are states of $G(w)$ and $x \sim y$ then $x, y \in \mathcal{I}_k$ for some $k \in \{0, 1, \ldots, K\}$.*

Proof. Because $G(w)$ is acyclic, contains exactly one accepting state $\emptyset$ and all transitions from layer $\mathcal{I}_k$ for $k \geq 1$ lead to layer $\mathcal{I}_{k-1}$, it can be proved by simple induction that all accepting paths starting from $x \in \mathcal{I}_k$ are of length k. By Observation 2 there exists at least one such path for every x. This concludes that equivalent states of $G(w)$ cannot belong to different layers. □

We will label all states of $G(w)$ layer by layer in the order $k = 0, 1, \ldots, K$ in such a way that equivalent states from a single layer receive equal labels.

Lemma 5. *The following labeling ℓ:*

- $\ell(\emptyset) = 0$
- $\ell(x) = (\ell(p), a, b, \ell(q))$ *for every state $x \in \mathcal{I}_k$ such that transitions in x are labeled with letters a and $\overline{b}$ and lead to states $p, q \in \mathcal{I}_{k-1}$ resp.*

preserves the equivalence of states.

Proof. It is a consequence of the definition of $G(w)$ and Observation 2. □

Unfortunately, the labels assigned as in Lemma 5 can be quite large. However, we can keep them of constant size if we renumber the quadruples in each layer with integers of size $O(n)$. This is always possible since each layer of $G(w)$ contains at most $2n$ states. The renumbering can be performed by radix sort in $O(n)$ time and space per each layer by using arrays of size $O(n)$ and $K = O(n)$ for dimensions 1, 4 and 2, 3 of the quadruples resp.

Let us summarize the whole discussion. Due to Lemma 3, the interval automaton can be constructed layer by layer in $O(n)$ time and space per each layer. We have described an algorithm for labeling of states of $G(w)$ that preserves equivalence of states, executes in the same ordering of layers as the algorithm from Lemma 3 and has the same complexity. Due to Lemmas 1 and 2, this implies the following result.

Theorem 2 (Main result).
There exists an algorithm for checking whether $u \approx v$ for two words of total length n in $O(n \cdot |\Sigma|)$ time and $O(n)$ space.

5 Final remarks

It can be observed that almost all labels of transitions in the automaton $G(w)$ can be removed without changing the output of the algorithm. More precisely, all labels from Σ apart from the transitions starting in layer 1 and all labels from $\overline{\Sigma}$ can be replaced by a special label $\# \notin \Sigma$, resulting in automaton $G'(w)$. It can be proved by a layer–by–layer induction that $u \sim v$ in $G'(w)$ iff $u \sim v$ in $G(w)$. Unfortunately, this does not lead to any improvement of the time complexity of the whole algorithm. We would like to thank Marcin Andrychowicz for showing us this observation.

We described a very efficient algorithm for testing if $u \approx v$ in a free idempotent semigroup. The remaining problem is to design an algorithm that transforms u to v replacing factors x^2 by x or z by z^2.

Lemma 6. *If $u, v \in \Sigma^*$, $|u| + |v| = O(n)$ and $u \approx v$ then there exists a sequence of "idempotent" transformations from u to v of length $O(2^{|\Sigma|}n)$.*

Proof. The proof of the Green–Rees theorem is constructive and the sequence of transformations it generates is of length $O(2^{|\Sigma|}n)$. □

The length of the sequence of steps from Lemma 6 is exponential in $|\Sigma|$. Thus the following open problems remain:

- Does there exist a polynomial time deterministic algorithm that always generates a sequence of transformations that is of polynomial length in terms of n and $|\Sigma|$?
- Does there exist such an algorithm for finding the *smallest* number of steps necessary to transform u to v?

References

1. Adjan, S.I.: The Burnside problem and identities in groups. In: Ergebnisse der Mathematik und ihrer Grenzgebiete 95 [Results in Mathematics and Related Areas]. Translated from Russian by John Lennox and James Wiegold. Springer-Verlag, Berlin-New York (1979)
2. Burnside, W.: On an unsettled question in the theory of discontinuous groups. Quart. J. Pure Appl. Math. 33, 230–238 (1902)
3. de Luca, A., Varricchio, S.: On non-counting regular classes. In: Paterson, M.S. (ed.) Automata, Languages and Programming. LNCS, vol. 443, pp. 74–87, Springer-Verlag, Berlin (1990)
4. de Luca, A., Varricchio, S.: On non-counting regular classes. Theoret. Comput. Sci. 100, 67–104 (1992)
5. do Lago, A.P.: On the Burnside semigroups $x^n = x^{n+m}$. In: Simon, I. (ed.) LATIN 1992. LNCS, vol. 583, pp. 329–343. Springer-Verlag, Berlin (1992)
6. do Lago, A.P.: On the Burnside semigroups $x^n = x^{n+m}$. Int. J. Algebra Comput. 6, 179–227 (1996)
7. do Lago, A.P., Simon, I.: Free Burnside Semigroups. Theoret. Informatics Appl. 35, 579–595 (2001)
8. Green, J.A., Rees, D.: On semigroups in which $x^r = x$. Math. Proc. Camb. Phil. Soc. 48, 35–40 (1952)
9. Guba, V.S.: The word problem for the relatively free semigroup satisfying $t^m = t^{m+n}$ with $m \geq 3$. Int. J. Algebra Comput. 2, 335–348 (1993)

10. Guba, V.S.: The word problem for the relatively free semigroup satisfying $t^m = t^{m+n}$ with $m \geq 4$ or $m = 3, n = 1$. Int. J. Algebra Comput. 2, 125–140 (1993)
11. Kaďourek, J., Polák, L.: On free semigroups satisfying $x^r = x$. Simon Stevin 64, 3–19 (1990)
12. Karhumaki, J.: Combinatorics on words. Notes in pdf
13. Lallement, G.: Semigroups and Combinatorial Applications. J. Wiley and Sons, New York (1979)
14. Lothaire, M.: Combinatorics on Words. Addison-Wesley, Reading, MA., U.S.A. (1983)
15. McCammond, J.: The solution to the word problem for the relatively free semigroups satisfying $t^a = t^{a+b}$ with $a \geq 6$. Int. J. Algebra Comput. 1, 1–32 (1991)
16. Plyushchenko, A.N., Shur, A.M.: Almost overlap-free words and the word problem for the free Burnside semigroup satisfying $x^2 = x^3$. Proc. of WORDS'07