Efficient Construction of Semilinear Representations of Languages Accepted by Unary NFA

Zdeněk Sawa⋆

Center for Applied Cybernetics, Department of Computer Science
Technical University of Ostrava
17. listopadu 15, Ostrava-Poruba, 708 33, Czech republic
zdenek.sawa@vsb.cz

Abstract. Chrobak (1986) proved that a language accepted by a given nondeterministic finite automaton with one-letter alphabet, i.e., a unary NFA, with n states can be represented as the union of $O(n^2)$ arithmetic progressions, and Martinez (2002) has shown how to compute these progressions in polynomial time. To (2009) has pointed out recently that Chrobak's construction and Martinez's algorithm, which is based on it, contain a subtle error and has shown how they can be corrected. In this paper, a new simpler and more efficient algorithm for the same problem is presented. The running time of the presented algorithm is $O(n^2(n+m))$, where n is the number of states and m the number of transitions of a given unary NFA.

1 Introduction

It is well known that Parikh images of regular (and even context-free) languages are semilinear sets [7, 4]. In *unary* languages, i.e., languages over a one-letter alphabet, words can be identified with their lengths (i.e., a^n can be identified with n), so the Parikh image of a unary language is just the set of lengths of words of the language, and it can be identified with the language itself. It can be easily shown that each regular unary language can be represented as the union of a finite number of arithmetic progressions of the form $\{c + di \mid i \in \mathbb{N}\}$ where c and d are constants specifying the offset and the period of a progression.

A unary nondeterministic finite automaton (a unary NFA) is an NFA with a one-letter alphabet. Given a unary NFA $\mathcal{A}$, a set of arithmetic progressions representing the language accepted by $\mathcal{A}$ can be computed by determinization of $\mathcal{A}$; however, this straightforward approach can produce an exponential number of progressions. Chrobak [1] has shown that this exponential blowup is avoidable and that a language accepted by a unary NFA with n states can be represented as the union of $O(n^2)$ progressions of the form $\{c + di \mid i \in \mathbb{N}\}$ where $c < p(n)$ for some $p(n) \in O(n^2)$ and $0 \leq d \leq n$. The computational complexity of the

⋆ Supported by the Czech Ministry of Education, Grant No. 1M0567.

construction of these progressions was not analyzed in [1], but it can be easily seen that a naive straightforward implementation would require exponential time. Later, Martinez [5, 6] has shown how the construction described in [1] can be realized in polynomial time. The exact complexity of Martinez's algorithm is $O(kn^4)$ where n is the number of states of the automaton and k the number of strongly connected components of its graph. The result was recently used for example in [3, 2] to obtain more efficient algorithms for some problems in automata theory and the verification of one-counter processes.

In [8], To pointed out that Chrobak's construction and Martinez's algorithm (whose correctness relies on correctness of Chrobak's construction) contain a subtle error, and he has shown modifications that correct this error.

In this paper, we give a simpler and more efficient algorithm for the same problem, i.e., for computing of a corresponding set of arithmetic progressions for a given unary NFA. The time complexity of the algorithm is $O(n^2(n+m))$ and its space complexity $O(n^2)$, where n is the number of states and m number of transitions of the unary NFA.

Section 2 gives basic definitions and formulates the main result, Section 3 describes the algorithm and proofs of its correctness, and Section 4 contains a description of an efficient implementation of the algorithm and an analysis of its complexity.

2 Definitions and Main Result

The set of natural numbers $\{0, 1, 2, \ldots\}$ is denoted by $\mathbb{N}$. For $i, j \in \mathbb{N}$ such that $i \leq j$, $[i, j]$ denotes the set $\{i, i+1, \ldots, j\}$, and $[i, j)$ denotes the (possibly empty) set $\{i, i+1, \ldots, j-1\}$. Given $c, d \in \mathbb{N}$, an *arithmetic progression* is the set $\{c + d \cdot i \mid i \in \mathbb{N}\}$, denoted $c + d\mathbb{N}$, where c is called the *offset* and d the *period* of the progression.

The following definitions are standard (see e.g. [4]), except that they are specialized to the case where a one-letter alphabet is used. In such an alphabet, words can be identified with their lengths.

A *unary nondeterministic finite automaton (a unary NFA)* is a tuple $\mathcal{A} = (Q, \delta, I, F)$ where Q is a finite set of *states*, $\delta \subseteq Q \times Q$ is a *transition relation*, and $I, F \subseteq Q$ are sets of *initial* and *final* states respectively. A *path* of length k from q to q', where $q, q' \in Q$, is a sequence of states $q_0, q_1, \ldots, q_k$ from Q where $q = q_0$, $q' = q_k$, and $(q_{i-1}, q_i) \in \delta$ for each $i \in [1, k]$. We use $q \xrightarrow{k} q'$ to denote that there exists a path of length k from q to q'. A word $x \in \mathbb{N}$ is *accepted* by $\mathcal{A}$ if $q_0 \xrightarrow{x} q_f$ for some $q_0 \in I$ and $q_f \in F$. The language $L(\mathcal{A})$ accepted by a unary NFA $\mathcal{A}$ is the set of all words accepted by $\mathcal{A}$.

We consider the following problem:

PROBLEM: UNFA-Arith-Progressions
INPUT: A unary NFA $\mathcal{A}$.
OUTPUT: A set $\{(c_1, d_1), (c_2, d_2), \ldots, (c_k, d_k)\}$ of pairs of natural numbers such that $L(\mathcal{A}) = \bigcup_{i=1}^{k}(c_i + d_i\mathbb{N})$.

The main result presented in this paper is:

Theorem 1. *There is an algorithm solving* UNFA-Arith-Progressions *with time complexity* $O(n^2(n+m))$ *and space complexity* $O(n^2)$ *where* n *is the number of states and* m *the number of transitions of a given unary NFA. The algorithm constructs* $O(n^2)$ *pairs of numbers and each constructed pair* (c_i, d_i) *satisfies* $c_i < 2n^2 + n$ *and* $d_i \leq n$.

3 $L(\mathcal{A})$ as Union of Arithmetic Progressions

In this section, we describe the algorithm for UNFA-Arith-Progressions and prove its correctness.

In the rest of the section, we assume a fixed unary NFA $\mathcal{A} = (Q, \delta, I, F)$ with $|Q| = n$.

3.1 The algorithm

The algorithm works as follows. It computes the resulting set $\mathcal{R}$ of pairs of numbers that represent arithmetic progressions as the union of the following sets $\mathcal{R}_1$ and $\mathcal{R}_2$ where:

- $\mathcal{R}_1$ is the set of all of pairs $(x, 0)$ where $x \in L(\mathcal{A})$ and $x \in [0, 2n^2 + n)$, and
- $\mathcal{R}_2$ is the set of all of pairs (c, d) where $d \in [1, n]$, $c \in [2n^2 - d, 2n^2)$, and where for some $q_0 \in I$, $q \in Q$, and $q_f \in F$ we have $q_0 \xrightarrow{n} q$, $q \xrightarrow{d} q$, and $q \xrightarrow{c-n} q_f$ (note that $c \geq n$).

To compute $\mathcal{R}_1$, it is sufficient to test for each $x \in [0, 2n^2+n)$ if $x \in L(\mathcal{A})$, and to compute $\mathcal{R}_2$, it is sufficient to test for each of $O(n^2)$ pairs (c, d), where $d \in [1, n]$ and $c \in [2n^2 - d, 2n^2)$, if the required conditions are satisfied. All these tests can be easily done in polynomial time and we can also see that $|\mathcal{R}| \in O(n^2)$. An efficient implementation of the algorithm, which avoids some recomputations by precomputing certain sets of states, is described in Section 4 together with a more detailed analysis of its complexity.

The correctness of the algorithm is ensured by the following crucial lemma and its corollary; the proof of the lemma is postponed to the next subsection.

Lemma 2. *Let* $x \geq 2n^2 + n$. *If* $x \in L(\mathcal{A})$ *then* $x \in c + d\mathbb{N}$ *for some* $(c, d) \in \mathcal{R}_2$.

Corollary 3. *Let* $x \in \mathbb{N}$. *Then* $x \in L(\mathcal{A})$ *iff* $x \in c + d\mathbb{N}$ *for some* $(c, d) \in \mathcal{R}$.

Proof. ($\Rightarrow$) Assume $x \in L(\mathcal{A})$. Either $x < 2n^2 + n$ and then $(x, 0) \in \mathcal{R}_1$ and $x \in (x + 0\mathbb{N}) = \{x\}$, or $x \geq 2n^2 + n$ and then $x \in c + d\mathbb{N}$ for some $(c, d) \in \mathcal{R}_2$ by Lemma 2.

($\Leftarrow$) It can be easily checked that $c + d\mathbb{N} \subseteq L(\mathcal{A})$ for each $(c, d) \in \mathcal{R}$. For $(c, d) \in \mathcal{R}_1$ this follows from the definition, and for $(c, d) \in \mathcal{R}_2$ from the observation that if $q_0 \xrightarrow{n} q$, $q \xrightarrow{d} q$, and $q \xrightarrow{c-n} q_f$ for some $q_0 \in I$, $q \in Q$, and $q_f \in F$ (where $c \geq n$), then $\mathcal{A}$ accepts each word from $c + d\mathbb{N}$. $\square$

3.2 Proof of Lemma 2

The rest of this section is devoted to the proof of Lemma 2, which is done by the following sequence of simple propositions.

The basic idea of the proof is that there exists a polynomial $p(n) \in O(n^2)$ such that if $q_1 \xrightarrow{x} q_2$ for some $q_1, q_2 \in Q$ and $x \geq p(n)$ then there is a path α of length x from q_1 to q_2 of the following form: α goes from q_1 to some state q by c_1 steps, then goes through a cycle of length $d \in [1, n]$ several times, and then goes from q to q_2 by c_2 steps. Obviously $x = c_1 + k \cdot d + c_2$ for some $k \in \mathbb{N}$, and it will be also ensured that $c_1 + c_2 < p(n)$.

Every path α of length x from q_1 to q_2 can be transformed into the described form by the following construction: we can decompose α into *elementary cycles*, i.e., cycles where no state is repeated, and a *simple path*, i.e., a path where no state is repeated, from q_1 to q_2. We can do this by repeatedly removing elementary cycles from α. Using this decomposition, we can construct a path of the required form by selecting one elementary cycle of some length, say d, and by repeatedly "cutting-out" some subsets of the remaining elementary cycles, such that the sums of lengths of cycles in these subsets are multiples of d, which means that they can be replaced with iterations of the selected cycle of length d.

However, when we "cut-out" cycles, we must be careful, because by cutting-out some cycles, some other cycles can become unreachable. An error of this kind was made by Chrobak in [1] as pointed out by To in [8].

To ensure that none of the cycles becomes unreachable, we divide elementary cycles into two categories — *removable* and *unremovable*. Only removable cycles will be cut-out, and it will be ensured that it is safe to remove any subset of removable cycles.

We say a sequence $\beta_0, \beta_1, \ldots, \beta_r$, where β_0 is a simple path from q_1 to q_2 and where $\beta_1, \beta_2, \ldots, \beta_r$ are elementary cycles, is *good* if for each $i \in [1, r]$ there is some $j \in [0, i)$, such that β_i and β_j share at least one state q. Note that from such good sequence we can construct a path from q_1 to q_2, whose length is the sum of lengths of all β_i, by starting with β_0 and repeatedly "pasting-in" $\beta_1, \beta_2, \ldots, \beta_r$ (in this order). Each cycle β_i can be "pasted-in" since it shares some state q with some β_j where $j < i$ (β_i can be pasted in by splitting it in q).

Note that a decomposition $\beta_0, \beta_1, \ldots, \beta_r$ of an original path α, where β_0 is a simple path from q_1 to q_2 and where $\beta_1, \beta_2, \ldots, \beta_r$ are elementary cycles in the reverse order, in which they were removed from α (i.e., β_r was removed first and β_1 last), is good. We say a cycle β_i, where $i \in [1, r]$, is *removable* if for each state q of β_i there is some $j \in [0, i)$ such that β_j contains q. Cycle β_i that is not removable is *unremovable*. It can be easily checked that a sequence obtained from $\beta_0, \beta_1, \ldots, \beta_r$ by removing some arbitrary subset of removable cycles is also good.

The following proposition is the main "tool" that allows us to find a subset of removable cycles such that the sum of lengths of cycles in this subset is a multiple of d.

Proposition 4. *Let $d \geq 1$. Every sequence $x_1, x_2, \ldots, x_r$ of natural numbers, where $r \geq d$, contains a non-empty subsequence $x_i, x_{i+1}, \ldots, x_j$ (where $1 \leq i \leq j \leq r$) such that $(x_i + x_{i+1} + \cdots + x_j) \equiv 0 \pmod{d}$.*

Proof. Consider a sequence $s_0, s_1, \ldots, s_r$ where $s_i = x_1+x_2+\cdots+x_i$ for $i \in [0, r]$. There are at most d different values of s_i modulo d. Since $r \geq d$, by the pigeonhole principle we have $s_i \equiv s_j \pmod{d}$ for some i, j such that $0 \leq i < j \leq r$. The nonempty sequence $x_{i+1}, x_{i+2}, \ldots, x_j$ has the required property $(x_{i+1} + x_{i+2} + \cdots + x_j) \equiv 0 \pmod{d}$, since $s_j - s_i \equiv 0 \pmod{d}$. $\square$

Proposition 5. *Let $q_1, q_2 \in Q$, $x \in \mathbb{N}$, and $d \in [1, n]$. If $q_1 \xrightarrow{x} q_2$ then $q_1 \xrightarrow{y} q_2$ for some $y \in [0, 2n^2 - n)$ such that $y \leq x$ and $y \equiv x \pmod{d}$.*

Proof. Let us assume $q_1 \xrightarrow{x} q_2$ and let $y \in \mathbb{N}$ be the smallest number such that $y \equiv x \pmod{d}$ and $q_1 \xrightarrow{y} q_2$ (such y exists, since $y = x$ satisfies these properties). Let $\beta_0, \beta_1, \ldots, \beta_r$ be a good decomposition of a path of length y from q_1 to q_2 (β_0 is a simple path from q_1 to q_2 and β_i for $i \in [1, r]$ are elementary cycles). Let us assume that there are at least d removable cycles in this decomposition. Then, by Proposition 4, there is a nonempty subset of these removable cycles such that the sum of lengths of the cycles in this subset is a multiple of d. By removing the cycles in this subset we obtain a good sequence, from which we can construct a path from q_1 to q_2 of length $y' < y$ where $y' \equiv y \pmod{d}$. So $q_1 \xrightarrow{y'} q_2$ and $y' \equiv x \pmod{d}$, which is a contradiction, since we have assumed that y is the smallest such number. This implies that in the sequence $\beta_0, \beta_1, \ldots, \beta_r$ there are at most $d - 1$ removable cycles.

A cycle β_i is unremovable iff it contains a state q that does not belong to any β_j with $j < i$, which implies that there are at most $n-1$ unremovable cycles (note that there is at least one state in β_0). The length of β_0 is at most $n - 1$ and a length of each elementary cycle is at most n, which implies

$$y \leq (n-1) + (n-1+d-1) \cdot n < 2n^2 - n,$$

since $d \leq n$. $\square$

Corollary 6. *Let $q_1 \xrightarrow{x} q_2$ for some $q_1, q_2 \in Q$ and $x \in \mathbb{N}$. If $x \geq n$ then there exist $q \in Q$, $c_1 \in [0, n)$, $d \in [1, n]$, and $c_2 \in [0, 2n^2 - n)$ such that $q_1 \xrightarrow{c_1} q$, $q \xrightarrow{d} q$, $q \xrightarrow{c_2} q_2$, and $x \in (c_1 + c_2) + d\mathbb{N}$.*

Proof. By the pigeonhole principle, some $q \in Q$ must be visited twice in the first n steps of a path from q_1 to q_2 of length $x \geq n$, and so for some $c_1 \in [0, n)$, $d \in [1, n]$, and $c_2' \in \mathbb{N}$ we have $q_1 \xrightarrow{c_1} q$, $q \xrightarrow{d} q$, $q \xrightarrow{c_2'} q_2$, and $x = c_1 + d + c_2'$. By Proposition 5, there is some $c_2 \in [0, 2n^2 - n)$ satisfying $c_2 \leq c_2'$, $q \xrightarrow{c_2} q_2$, and $c_2 \equiv c_2' \pmod{d}$. So $c_2' = c_2 + k \cdot d$ for some $k \in \mathbb{N}$, and $x = c_1 + d + c_2' = (c_1 + c_2) + (k+1) \cdot d$, which means that $x \in (c_1 + c_2) + d\mathbb{N}$. $\square$

Proposition 7. *Let $q_1 \xrightarrow{x} q_2$ for some $q_1, q_2 \in Q$ and $x \in \mathbb{N}$. If $x \geq 2n^2 + n$ then there exist $q \in Q$, $c \in [0, 2n^2 - n)$, and $d \in [1, n]$, such that $q_1 \xrightarrow{n} q$, $q \xrightarrow{d} q$, $q \xrightarrow{c} q_2$, and $x \in (n + c) + d\mathbb{N}$.*

Proof. Assume $q_1 \xrightarrow{x} q_2$ where $x \geq 2n^2 + n$. By Corollary 6, there are some $q' \in Q$, $c_1 \in [0, n)$, $d \in [1, n]$, $c_2 \in [0, 2n^2 - n)$, and $k \in \mathbb{N}$ such that $q_1 \xrightarrow{c_1} q'$, $q' \xrightarrow{d} q'$, $q' \xrightarrow{c_2} q_2$, and $x = (c_1 + c_2) + k \cdot d$. Let α be a path of length x from q_1 to q_2 that goes from q_1 to q' by c_1 steps, then goes k times through a cycle β of length d, and then goes from q' to q_2 by c_2 steps, and let q be the state reached after the first n steps of α. Note that since $(c_1 + c_2) + k \cdot d = x \geq 2n^2 + n$ and $c_1 + c_2 < 2n^2$ (because $c_1 < n$ and $c_2 < 2n^2 - n$), we have $k \cdot d \geq n$. Together with $c_1 < n$ this ensures that the state q is on the cycle β, which implies $q_1 \xrightarrow{n} q$, $q \xrightarrow{d} q$, and $q \xrightarrow{x-n} q_2$. By Proposition 5, there is some $c \in [0, 2n^2 - n)$ such that $c \leq x - n$, $q \xrightarrow{c} q_2$, and $c \equiv x - n \pmod{d}$. This means that $n + c \equiv x \pmod{d}$, and since $c \leq x - n$ implies $n + c \leq x$, we have $x \in (n + c) + d\mathbb{N}$. ☐

Now we can prove Lemma 2.

Proof (of Lemma 2). Assume that $x \geq 2n^2 + n$ and $x \in L(\mathcal{A})$, so there are some $q_0 \in I$ and $q_f \in F$ such that $q_0 \xrightarrow{x} q_f$. By Lemma 7, there exist $q \in Q$, $c' \in [0, 2n^2 - n)$, and $d \in [1, n]$, such that $q_0 \xrightarrow{n} q$ $q \xrightarrow{d} q$, $q \xrightarrow{c'} q_f$, and $x \in (n + c') + d\mathbb{N}$. This means that for each $c \in (n + c') + d\mathbb{N}$, such that $c \leq x$, we have $q \xrightarrow{c-n} q_f$ and $x \in c + d\mathbb{N}$. In particular, there is one such c in the interval $[2n^2 - d, 2n^2)$, since $n + c' \in [n, 2n^2)$. ☐

4 Efficient Implementation

To avoid recomputations, the algorithm precomputes some sets. For $i \in \mathbb{N}$ we define $S_i = \{q \in Q \mid \exists q_0 \in I : q_0 \xrightarrow{i} q\}$ and $T_i = \{q \in Q \mid \exists q_f \in F : q \xrightarrow{i} q_f\}$, and for $q \in Q$ we define $\mathit{Periods}(q) = \{d \in [1, n] \mid q \xrightarrow{d} q\}$. In particular, the algorithm precomputes the sets S_n, T_i for $i \in [2n^2 - 2n, 2n^2 - n)$, and $\mathit{Periods}(q)$ for $q \in S_n$. To test for a given q if $q_0 \xrightarrow{n} q$ for some $q_0 \in I$, the algorithm tests if $q \in S_n$, to test if $q \xrightarrow{c-n} q_f$ for some $q_f \in F$, it tests if $q \in T_{c-n}$, and to test if $q \xrightarrow{d} q$, it tests if $d \in \mathit{Periods}(q)$.

All these sets can be implemented as bit arrays, so operations like adding an element to a set, testing if an element is member of a set, and so on, can be performed in a constant time. It is also obvious that for $Q' \subseteq Q$, the sets $\mathit{Succ}(Q') = \{q \in Q \mid \exists q' \in Q' : (q', q) \in \delta\}$ and $\mathit{Pre}(Q') = \{q \in Q \mid \exists q' \in Q' : (q, q') \in \delta\}$ can be computed in time $O(n + m)$ where m is the number of transitions (i.e., $|\delta| = m$). Using subroutines for computing *Pre* and *Succ*, the precomputation of all necessary sets can be done in time $O(n^2(n + m))$. For example, S_n can be precomputed by computing sequence $S_0, S_1, \ldots, S_n$ where $S_0 = I$, and $S_{i+1} = \mathit{Succ}(S_i)$ for $i \geq 0$, T_i can be computed by $T_0 = F$, and $T_{i+1} = \mathit{Pre}(T_i)$ for $i \geq 0$, etc. Also all $x < 2n^2 + n$ such that $x \in L(\mathcal{A})$ can be found in time $O(n^2(n + m))$ by computing the sequence $S_0, S_1, \ldots, S_{2n^2+n-1}$ and checking if $S_x \cap F \neq \emptyset$ for $x \in [0, 2n^2 + n)$.

There are $O(n^2)$ pairs (c, d) such that $d \in [1, n]$ and $c \in [2n^2 - d, 2n^2)$, and for each of them, at most n states are tested. Since the corresponding tests for one triple c, d, q can be done in a constant time as described above, all triples can be tested in time $O(n^3)$. We see that the overall running time of the algorithm is $O(n^2(n+m))$.

During the computation, only the values of S_n, T_i for $i \in [2n^2-n-d, 2n^2-n)$, and $Periods(q)$ for $q \in S_n$ need to be stored. Obviously, $O(n^2)$ bits are sufficient to store these values. Other values are used only temporarily, can be discarded after their use, and do not take more than $O(n^2)$ bits, so the overall space complexity of the algorithm is $O(n^2)$.

References

1. Chrobak, M.: Finite automata and unary languages. Theoretical Computer Science 47(2), 149–158 (1986)
2. Göller, S., Mayr, R., To, A.W.: On the computational complexity of verifying one-counter processes. In: LICS'09. pp. 235–244. IEEE Computer Society (2009), http://dx.doi.org/10.1109/LICS.2009.37
3. Gruber, H., Holzer, M.: Computational complexity of NFA minimization for finite and unary languages. In: LATA'08. Lecture Notes in Computer Science, vol. 5196, pp. 261–272. Springer (2008)
4. Kozen, D.C.: Automata and Computability. Springer-Verlag (1997)
5. Martinez, A.: Efficient computation of regular expressions from unary nfas. In: Descriptional Complexity of Formal Systems (DFCS) (2002)
6. Martinez, A.: Topics in Formal Languages: String Enumeration, Unary NFAs and State Complexity. Master's thesis, University of Waterloo (2002)
7. Parikh, R.J.: On context-free languages. J. ACM 13(4), 570–581 (1966)
8. To, A.W.: Unary finite automata vs. arithmetic progressions. Information Processing Letterss 109(17), 1010–1014 (2009), http://dx.doi.org/10.1016/j.ipl.2009.06.005