The World of Unary Languages.*
A Quick Tour

Carlo Mereghetti
Dip. di Informatica, Sist. e Com.
Univ. degli Studi di Milano – Bicocca
via Bicocca degli Arcimboldi 8
20126 Milano, Italy
mereghetti@disco.unimib.it

Giovanni Pighizzini
Dip. di Scienze dell’Informazione
Univ. degli Studi di Milano
via Comelico 39
20135 Milano, Italy
pighizzi@dsi.unimi.it

Abstract
We give two flashes from the world of unary languages related to the study of tight computational lower bounds for nonregular language acceptance. They show both an interesting dissymmetry with the general case, and the flavor and some typical number theoretic tools of unary computations.

1 Introduction

In mathematics, “simplifications” often lead to meaningful and interesting results. This is exactly the case even for theoretical computer science and, particularly, for language and complexity theory. In such realms, one of the most investigated simplification is that provided by unary languages. A unary language is simply a language built over a single–letter input alphabet. Several results in the literature witness the relevance of dealing with unary inputs, often emphasizing sharp dissymmetries with the general case of languages on alphabets with two or more symbols.

However, it is worth remarking that simplifications do not necessarily lead to easier problems to cope with, and this is the case even when considering unary languages. The total absence of structure in input strings forces our computation techniques to be much more sophisticated and sometimes tricky. In particular, as one may expect, it is interesting to record the constant and nice use of Number Theory which provides a valuable source of nontrivial tools and properties to operate with.

In this work we are going to briefly survey some results showing the importance and the typical flavor of computing with unary languages. Specifically, we focus on the world of Turing machines that work within

---

*Partially supported by MURST, under the project “Modelli di calcolo innovativi: metodi sintattici e combinatori”.
a sublogarithmic amount of space. Here, several contributions emphasize the importance of investigating unary computations. It is well-known, for instance, that the truth of both Immerman–Szelepcsényi and Savitch's Theorems remains an open problem when considering sublogarithmic work space. By contrast, Viliam Geffert [5] proves that these two theorems hold true for unary languages processed in sublogarithmic space. Beside this, other examples witnessing the relevance of considering the unary version of problems can be found, e.g., in [3, 6, 13].

Indeed, computing within sublogarithmic space turns out to be quite difficult since many traditional programming tools are missing. For instance, in such a restricted amount of tape, we cannot store input length, or fix positions on the input tape. Yet, difficulties get even worse when considering inputs that are completely structureless, such as strings in unary languages. As a result, algorithms and proofs are often highly involved, typically relying on number theoretic issues (see, e.g., [6]).

In what follows, we are to deal with two instances of a lower bound problem that is tightly connected to the sublogarithmic space world, and for which considering unary languages turns out to be very fruitful. Namely, we refer to the following question first posed in 1965 by Hartmanis, Stearns, and Lewis [9, 11]: “What is the minimal amount of space used by a Turing machine — equipped with a read–only input tape and a read/write work tape — that accepts a nonregular language?”

This question has been extensively answered for deterministic, non-deterministic, and alternating Turing machines, and for different ways of measuring space (see [3, 13]). Furthermore, a generalization asking for lower bounds on both space and number of input head reversals has also been studied [2, 3]. Here, we concentrate on two cases.

The first one, presented in Section 3 and studied in [10], exhibits a situation in which the investigation on unary and general languages gives two different results. It concerns one–way (i.e., input is scanned once, from left to right) alternating Turing machines that work in middle space $s(n)$: all computations on all accepted inputs of length $n$ use no more than $s(n)$ work tape cells. What is the lower bound on $s(n)$ when the recognized language is nonregular? For nonregular languages built over general alphabets, the optimal lower bound is known to be $s(n) \notin o(\log \log n)$ [12]. By contrast, for unary nonregular languages, we show that the optimal lower bound is strictly higher, namely $s(n) \notin o(\log n)$.

The second one, presented in Section 4 and studied in [2], enables us to appreciate both the hardness of accepting unary languages within a
very restricted amount of computational resources, and the great help we can gain from Number Theory. It deals with the optimality proof of the lower bound on the product space $\times$ number of input head reversals, write $s(n) \cdot i(n)$, for nondeterministic Turing machines accepting nonregular languages in strong $s(n)$ space and $i(n)$ input head reversals: all computations on inputs of length $n$ use both no more than $s(n)$ work tape cells, and no more than $i(n)$ input head reversals. In [2], it is proved that $s(n) \cdot i(n) \notin o(\log n)$. It should be stressed that, without imposing restrictions on $i(n)$, so to have general two–way nondeterministic Turing machines, a lower bound of strong $s(n) \notin o(\log \log n)$ is given in [8]. Thus, it would be interesting to give the optimality of $s(n) \cdot i(n) \notin o(\log n)$ in the “best possible way”, i.e., by exhibiting a unary nonregular language accepted by a nondeterministic Turing machine in strong $O(\log \log n)$ space and $O(\log n/\log \log n)$ input head reversals. This is exactly what we are going to do by considering a unary nonregular language proposed by Alt and Mehlhorn in [1], namely, $L_{AM} = \{1^n \mid q(n) \text{ is a power of 2}\}$, where $q(n)$ denotes the smallest integer not dividing $n$. We will show how a very efficient recognizing algorithm for $L_{AM}$ can be designed by studying some number theoretic properties of $q(n)$.

Before taking a look into the unary world, we need some technical details recalled in the next section.

2 Preliminary notions

Let $\Sigma^*$ the set of all strings (with the empty string $\varepsilon$) on an alphabet $\Sigma$. We let $|x|$ be the length of $x \in \Sigma^*$. A language $L \subseteq \Sigma^*$ is unary (or tally), whenever $\Sigma$ is a single–letter alphabet. In this case, we let $\Sigma = \{1\}$.

We consider the standard Turing machine model (see [9, 11, 13]) having a finite state control, a two–way read–only input tape (with input enclosed between a left and a right endmarker symbol, ‘$\not c$’ and ‘$\not c$', respectively), and a separate semi–infinite two–way read–write work tape. A memory state$^1$ of a Turing machine $M$ is an ordered triple $(q, w, i)$ where $q$ is a finite control state, the string $w$ is the nonblank content of the work tape, and the integer $1 \leq i \leq |w| + 1$ denotes the head position on the work tape. A configuration of $M$ on a given input string $x$ is an ordered pair $(j, m)$ where the integer $0 \leq j \leq |x| + 1$ represents the input head position, while $m$ is a memory state. The initial configuration is

$^1$Sometimes called internal configuration (see, e.g., [12, 13]).
the pair \((0, (q_0, \varepsilon, 1))\) where \(q_0\) is the initial state. An accepting configuration is any configuration containing a final state. It is well–known that a computation of a deterministic or nondeterministic Turing machine on a given input string can be seen as a finite or infinite sequence \(c_0, c_1, \ldots, c_i, \ldots\) of configurations beginning in the initial configuration \(c_0\) and such that \(c_{i+1}\) is an immediate successor of \(c_i\) on the appropriate input symbol. A computation is accepting whenever ending in an accepting configuration.

The space used in a computation is the maximum number of work tape cells taken by the configurations in that computation. An input head reversal in a computation is a sequence \(\alpha, \beta_1, \beta_2, \ldots, \beta_s, \gamma\) (\(s \geq 1\)) of configurations each one being the immediate successor of the previous one, and having the form \(\alpha = (j - 1, m')\), \(\beta_i = (j, m''_i)\), \(\gamma = (j - 1, m'''_i)\), or also \(\alpha = (j + 1, m')\), \(\beta_i = (j, m''_i)\), \(\gamma = (j + 1, m'''_i)\). A Turing machine is said to be one–way whenever its computations never present input head reversals, otherwise it is two–way.

The reader is assumed to be familiar with the notion of alternating Turing machine [4]. We only recall that, in an alternating Turing machine, the finite control states — and hence configurations and memory states as well — are partitioned into existential and universal. A computation is described by a tree whose nodes are labelled by configurations. The root is labelled by the initial configuration. Any internal node labelled by an existential configuration \(e\) has a unique son labelled by an immediate successor of \(e\). Any internal node labelled by a universal configuration \(u\) has a son for each immediate successor of \(u\). A computation is accepting if the corresponding tree is finite and all its leaves are labelled by accepting configurations.

The space used in a computation of an alternating Turing machine is the maximum number of work tape cells taken by the configurations labelling the corresponding tree. The number of input head reversals used in that computation is the maximum number of input head reversals along computation paths from the root.

Several notions of space complexity have been defined in the literature [3, 13]. Here, we are interested in strong [8, 9, 11] and middle [12] space. A Turing machine works in strong (middle) \(s(n)\) space if any computation on each (accepted) input of length \(n\) uses no more than \(s(n)\) space. These two space notions (and the others proposed in the literature) coincide for fully space constructible\(^2\) bounds, e.g., “normal”

\(^2\)A function \(s(n)\) is said to be fully space constructible if there exists a deterministic
functions above $\log n$. We are also interested in Turing machines simultaneously bounded on both space and input head reversals [2, 3]. Precisely, we say that a Turing machine, works in strong (middle) $s(n)$ space and $i(n)$ input head reversals if any computation on each (accepted) input of length $n$ uses no more than $s(n)$ space and $i(n)$ input head reversals.

3 A gap result for one–way middle alternation: when unary differs from general

What is the minimal middle space complexity $s(n)$ for a one–way alternating Turing machine that accepts a nonregular language? The answer to this question has been given by Szepietowski [12]: $s(n) \notin o(\log \log n)$. This lower bound is tight. In fact, in [12], a nonregular language is exhibited that is recognized by a one–way alternating Turing machine in middle $O(\log \log n)$ space. Such a language is defined as

$$L_S = \{ a^kb^m | m \text{ is a common multiple of all } r \leq k \}.$$

It is interesting to have an idea of how a one–way alternating Turing machine for $L_S$ works. We will have the first occasion to see both a very space–inexpensive computation, and Number Theory at work!

We begin by studying the structure of the strings in $L_S$. To this aim, we need a well–known result proved by P. Čebyšev in 1851. Let the function $\pi(x) =$ number of primes not exceeding $x$. Čebyšev’s Theorem (see, e.g., [7]) states that $c_1 x/\ln x \leq \pi(x) \leq c_2 x/\ln x$, for two positive constants $c_1$ and $c_2$, and with “ln” denoting the natural logarithm.

This enables us to give a lower bound for the function $P(x) = \prod_{p \leq x} p$, where $p$ denotes a prime number. In fact, by Čebyšev’s Theorem, we can write $P(x) \geq (c_1 x/\ln x)! \geq 2^{\frac{c_1}{2} x}$, where the last inequality holds for sufficiently large $x$. This shows that

**Lemma 3.1** There exists a positive constant $d$ such that $P(x) \geq d^x$.

At this point, we can state the following

**Theorem 3.1** Let $a^kb^m$ belong to $L_S$. Then $k \in O(\log m)$.

**Proof.** Since any integer less than or equal to $k$ must divide $m$, then any prime not exceeding $k$ divides $m$ as well. This clearly implies that $m \geq P(k)$. By applying Lemma 3.1, the claimed result follows.

Turing machine which, on any input of length $n$, uses exactly $s(n)$ space.
With these numerical properties in our hands, we are now ready to exhibit a one–way alternating Turing machine $M$ accepting $L$ in middle $O(\log \log n)$ space. On input the string $x$, our machine $M$ first writes on its work tape, in binary notation, the number $k$ of $a$’s at the beginning of $x$. Then, it universally branches on each $r \leq k$, checking whether $r$ divides the number $m$ of $b$’s at the end of $x$. This latter operation can be accomplished by simply counting $m$ modulo $r$.

That $M$ is one–way comes straightforwardly. For space requirement, we observe that, if $x$ belongs to $L$, than the space used by $M$ is $O(\log k)$, namely, the amount of tape needed to store the number of $a$’s. But Theorem 3.1 states that $k \in O(\log m)$. Thus, we conclude that $s(|x|) \in O(\log \log m) \in O(\log \log |x|)$. Furthermore, we notice that if $x$ was not in $L$, then $M$ might use more than $\log \log |x|$ tape before rejecting. All this shows that the space is middle.

Needless to remark, the language $L$ is built over a binary alphabet. So the question arises of whether the middle space lower bound $s(n) \not\in o(\log \log n)$ can be “certified” even in the unary world. Stated in other words: Are we able to exhibit a unary nonregular language accepted in middle $O(\log \log n)$ space by a one–way alternating Turing machine? A negative answer to this question is contained in the following theorem stating an exponential gap between the general and the unary case:

**Theorem 3.2** Let $M$ be a one–way alternating Turing machine accepting a unary nonregular language $L$ in middle $s(n)$ space. Then $s(n) \not\in o(\log n)$. This bound is tight.

**Proof.** By contradiction, we assume $s(n) \in o(\log n)$. It is easy to see that for any integer $n$ such that the string $1^n$ belongs to $L$, the number of memory states of $M$ on input $1^n$ is bounded above by $2^{h \cdot s(n)}$, for a suitable positive constant $h$. Moreover, by our initial assumption on $s(n)$ and since $L$ is nonregular, there must exist an integer $n'$ such that the string $1^{n'}$ belongs to $L$ and $2^{h \cdot s(n')} < n'$. Let us prove the following

**Claim 3.1** Any memory state reachable by $M$ without scanning the right endmarker symbol can be reached in less than $n'$ moves.

**Proof.** It is enough to show that every memory state $m$ reachable by $M$ in $k \geq n'$ moves by reading 1’s on the input tape can be reached in $k' < k$ moves as well. Thus, let $m_0, m_1, \ldots, m_k$ be a sequence of memory states of $M$ such that $m_0$ is the initial memory state, $m_k = m$, and $m_{i+1}$ is an immediate successor of $m_i$, for each $0 \leq i \leq k - 1$.  

Consider the initial segment $S = m_0, m_1, \ldots, m_{n'}$. Each memory state in $S$ can be reached on input $1^{n'}$. Moreover, since $1^{n'}$ belongs to $L$ and $M$ is middle space bounded, then each memory state in $S$ takes at most $s(n')$ work tape cells. Recall that $2^{h \cdot s(n')} < n'$. Hence, a simple pigeonhole argument shows the existence of $0 \leq i < j \leq n'$ satisfying $m_i = m_j$. Thus, the sequence $m_0, m_1, \ldots, m_i, m_{j+1}, \ldots, m_k = m$ leads to $m$ in $k' = k + i - j$ moves.

An easy consequence of this Claim is that the computations of $M$ on 1’s of any input must go only through memory states requiring no more than $s(n')$ work tape cells. This enables us to exhibit a one–way alternating automaton$^3$ $A$ that accepts the language $L$.

Informally, $A$ has the set $M$ of all the memory states of $M$ using at most $s(n')$ work tape cells as set of states. Its transition function $\delta$ is the “immediate successor” relation on the memory states assumed by $M$ on reading 1’s. Our Claim ensures that $\delta(c, 1)$ is well defined for any $m$ in $M$. It is not hard to verify that both $A$ and $M$ exhibit the same behavior on 1’s of any input string. Furthermore, the part of the computations of $M$ taking place on the right endmarker symbol may be correctly resumed in $A$ by choosing final states as follows: each $m \in M$ is a final state if and only if $M$ accepts by starting from $m$ and scanning only the symbol ‘$’.

A well–known result in [4] states that one–way alternating automata exactly characterize the class of regular languages. This proves that $L$ is regular, and contradicts our initial assumption.

The optimality of our new lower bound for unary nonregular languages is easily provable since we are given a “wide” (logarithmic) amount of work space where to store, e.g., the length of the input by one input scan. Thus, for instance, the language $L_{AM}$ mentioned in Section 1 — for which we will exhibit very space–efficient recognizing algorithms in Section 4 — is easily seen to be recognized in $O(\log n)$ space even by a one–way strong deterministic Turing machine.

4 Number Theory for optimality

Let us come to our second lower bound investigation. The question now sounds like: What is the minimal amount of strong $s(n)$ space and $i(n)$

$^3$We recall that a one–way alternating automaton is a one–way finite state automaton where we distinguish between existential and universal states [4].
input head reversals for a nondeterministic Turing machine that accepts a nonregular language? The answer is given in [2]: \( s(n) \cdot i(n) \not\in o(\log n) \). This lower bound has to be compared with that proved in [8] which says that a two-way nondeterministic Turing machines that accepts a nonregular language in strong \( s(n) \) space must satisfy \( s(n) \not\in o(\log \log n) \).

The comparison of these two lower bounds challenges us to prove the optimality of \( s(n) \cdot i(n) \not\in o(\log n) \) in the “best possible way”, namely: Can we exhibit a nonregular language which is unary, and is accepted by a nondeterministic Turing machine performing \( O(\log n/\log \log n) \) input head reversals in strong \( O(\log \log n) \) space? The answer is positive. Actually we will do even better, since the unary nonregular language we are going to consider will be accepted within the prescribed resource bounds by a deterministic machine.

So, let us take a look to the unary nonregular language \( L_{AM} \) proposed by Alt and Mehlhorn in [1]. For any positive integer \( n \), let \( q(n) \) denote the smallest integer not dividing \( n \). The language is defined as

\[
L_{AM} = \{1^n \mid q(n) \text{ is a power of } 2\}.
\]

As for the language \( L_s \) in the previous section, we need some number theoretic tools to analyze the structure of the strings in \( L_{AM} \), and to evaluate the complexity of the algorithms for \( L_{AM} \). We begin with

**Theorem 4.1** \( q(n) \in O(\log n) \).

**Proof.** Since \( q(n) \) is the smallest integer not dividing \( n \), then each prime less than \( q(n) \) must divide \( n \). This clearly implies that \( n \geq P(q(n) - 1) \), whence the result follows from Lemma 3.1.

Theorem 4.1 enables us to estimate space and input head reversals of the following algorithm for \( L_{AM} \) whose correctness follows trivially:

```
input(1^n)
k := 2
while n \equiv 0 \mod k do /* at the end of this loop k will contain q(n) */
    k := k + 1
if k is a power of 2 then accept else reject
```

Such an algorithm can be easily implemented on a deterministic Turing machine \( M \) whose space requirement, on any input \( 1^n \), is basically that for storing the counter \( k \). Since the greatest value assumed by \( k \) is exactly \( q(n) \), we get that \( M \) uses strong \( s(n) = O(\log q(n)) \) space which,
in the light of Theorem 4.1, becomes $s(n) \in O(\log \log n)$. So, for what concerns the space, we have reached our goal.

Unfortunately, problems come with input head reversals. In fact, $M$ performs an input head reversals for each new value of $k$, up to $q(n)$. On each input traversal, $M$ checks the loop condition “$n \equiv 0 \pmod{k}$" by simply counting $n$ modulo $k$ (no added space is needed). Hence, we get $i(n) = q(n)$ that, by Theorem 4.1, reads as $i(n) \in O(\log n)$. To save some head reversals, we must improve our knowledge on $q(n)$:

\textbf{Lemma 4.1} For any positive integer $n$, $q(n)$ is a prime power.

\textit{Proof.} Assume that $q(n)$ is not a prime power. Thus, $q(n) = a \cdot b$, for two coprime integers $1 < a < b < q(n)$. Since $q(n)$ is the smallest integer not dividing $n$, then both $a$ and $b$ must divide $n$, and since they are coprime, their product $a \cdot b = q(n)$ divides $n$ as well, a contradiction. \hfill \blacksquare

This result immediately leads to the following improved algorithm for $\mathcal{L}_\text{AM}$, where $pp(x)$ denotes the predicate “$x$ is a prime power”:

\begin{verbatim}
input(1^n)
k := 2
while $n \equiv 0 \pmod{k}$ do
    begin
        $k := k + 1$
        while not($pp(k)$) do $k := k + 1$  /* skip non–prime powers */
    end
if $k$ is a power of 2 then \textbf{ACCEPT} else \textbf{REJECT}
\end{verbatim}

The space requirement is still seen to be $O(\log \log n)$ (in particular, it is not hard to argue that testing $pp(x)$ needs no extra space). What is changed is that now $i(n)$ equals the number of prime powers not exceeding $q(n)$. To compute this new value of $i(n)$, let $\Pi(x) =$ number of prime powers not exceeding $x$. We are to estimate the growth of $\Pi(x)$. To this purpose, we make use of the celebrated Prime Number Theorem due to J. Hadamard and C. de la Vallée Poussin in 1896 (see, e.g., [7]). Such a theorem refines that of Čebyšev by stating that $\pi(x) \sim x / \ln x$.

\textbf{Theorem 4.2} $\Pi(x) \sim \frac{x}{\ln x}$.

\textit{Proof.} For $1 \leq i \leq \pi(x)$, let $N_i$ be the number of powers of the $i$–th prime $p_i$ that do not exceed $x$. Hence, $\Pi(x) = \sum_{i=1}^{\pi(x)} N_i$. It is easy to
see that \( x \geq p_i^{N_i} \), and hence \( N_i \leq \ln x / \ln p_i \). By these facts, and using the Prime Number Theorem, we can write

\[
\frac{x}{\ln x} \sim \pi(x) \leq \Pi(x) \leq \sum_{i=2}^{\pi(x)} \frac{\ln x}{\ln p_i} \leq \sum_{i=2}^{\pi(x)} \frac{\ln x}{\ln i} \sim \ln x \int_{2}^{\pi(x)} \frac{1}{\ln t} dt \sim \frac{x}{\ln x}.
\]

Summing up, we had the improved algorithm for \( L_{AM} \) performing \( i(n) = \Pi(q(n)) \) input head reversals. By both Theorem 4.2 and Theorem 4.1, we obtain \( i(n) \in O(\log n / \log \log n) \) which concludes the story:

**Theorem 4.3** \( L_{AM} \) is recognized by a deterministic Turing machine in strong \( O(\log \log n) \) space with \( O(\log n / \log \log n) \) input head reversals.

**References**


