The Power of One-State Turing Machines

Marzio De Biasi

Jan 15, 2018

Abstract

At first glance, one–state Turing machines are very weak: the Halting problem for them is decidable, and, without memory, they cannot even accept a simple one element language such as $L = \{1\}$. Nevertheless it has been showed that a one-state Turing machine can accept non regular languages. We extend such result and prove that they can also recognize non context–free languages, so for some tasks they are more powerful than a pushdown automata.

1 Introduction

A Turing machine with two states is able to simulate any Turing machine; hence we can build a two–states Universal Turing machine (actually 2 states and 18 symbols are enough to achieve universality [4]) hence the Halting problem for them is undecidable. We dramatically clips their wings if we allow only one state: we get a device with no internal memory and the Halting problem becomes decidable [1, 5]. We cannot even distinguish between accepting and non–accepting states. Nevertheless we can relax the notion of acceptance and rejection of an input: a one–state Turing machine accepts a string x if it halts; it rejects x if it runs forever. Under this different notion of acceptance, a one–state Turing machine is more powerful than a finite state automata [2].

We extend such result and prove that a one–state Turing machine is more powerful than a pushdown automata, too: it can recognize non context–free languages.

In Section 2 we give some preliminary definitions, in Section 3 we build a one–state Turing machine that is able to *compare* two numbers in different bases, in Section 4 we give a formal proof that the language recognized by such machine is not context–free.

2 Definitions

We use the standard definition of Turing machine as a 7-tuple [6]:

Definition 2.1 (Turing Machine)**.** A Turing machine M is a 7-tuple $\langle Q, \Sigma, \Gamma, q_0, q_a, q_r, \delta \rangle$ where Q is the set of states, Σ is the input alphabet not containing the *blank symbol* $\sqcup$, Γ is the tape alphabet, $\sqcup \in \Gamma, \Sigma \subseteq \Gamma$, $q_0 \in Q$ is the initial state, $q_a \in Q$ is the accept halting state, $q_r \in Q$ is the reject halting state ($q_r \neq q_a$), $\delta : Q \setminus \{q_a, q_r\} \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function that given a non-halting state and the current symbol returns a new (possibly halting) state, a symbol that is written on the tape and the direction of the move ($Left$ or $Right$).

The single *head* of the Turing machine is placed on a *tape* with an infinite number of *cells* in both directions; at the beginning of the computation the state of the Turing machine is q_0, the head is placed on the leftmost symbol of the input string; the rest of the cells contain the blank symbol $\sqcup$.

With two states and 18 symbols ($|Q| = 2, |\Gamma| = 18$) we can build an Universal Turing machine [4]; we restrict our study to one–state Turing machines so $|Q| = 1$ [2, 5]. But with only one state available we cannot neither distinguish between an accepting and rejecting state nor between a non–halting and a halting state; so we must somewhat alter the standard definition of accept/reject:

Definition 2.2 (One–state Turing machine)**.** A one state Turing machine is a 6-tuple:
$\langle \{q\}, \Sigma, \Gamma, \mathrm{q}, H, \delta \rangle$ where q is the only state which is also the initial state, Σ is the input alphabet not containing the *blank symbol* $\sqcup$, Γ is the tape alphabet, $\sqcup \in \Gamma, \Gamma \subseteq \Sigma$, H is the set of *halting symbols*, $\delta : \Gamma \setminus H \to \Gamma \times \{L, R\}$ is the transition function that given the current symbol under the head, returns the new symbol to be written and the direction of the move. If the current symbol is a halting symbol, the machine halts.

We define the language recognized by a one–state Turing machine:

Definition 2.3 (Language recognized by a one–state Turing machine)**.** If M is a one–state Turing machine, $x \in \Sigma^*$, we define the language recognized by M the set:

$$L(M) = \{x \mid M \text{ halts on input x}\}$$

.

We could also consider the variant in which the one–state Turing machine is equipped with distinct sets of *accept* and *reject* halting symbols. In this case the behavior can be: halt and accept, halt and reject or run forever. However it is straightforward to see that the results would also extend to this model.

At first glance, a one–state Turing machine is very weak, indeed it cannot even recognize trivial regular languages:

Theorem 2.4. *No one–state Turing machine M can recognize the trivial regular language $L = \{1\}$.*

Proof. Suppose that M recognizes $L = \{1\}$. At the beginning of the computation the head is placed on the symbol 1, if it is a halting symbol then M halts, but it will also halt and accept on any string $x = 1^+$. If it is not a halting symbol, $(1, L) \in \delta$ or $(1, R) \in \delta$. Suppose that it moves Left; now the head is on a blank symbol; if it is halting, then again it will also halt and accept on any $x = 1^+$; so it must move back to the right, i.e. $(\sqcup, R) \in \delta$. But in this case it must accept without moving to the blank symbol to the right of the 1, which would cause an infinite sequences of right moves; so it must also accept all $x = 1^+$. So $(1, R) \in \delta$ and the first step is a right step. If the blank symbol is halting, then it would also accept all $x = 1^+$; so it must move back to the left, i.e. $(\sqcup, L) \in \delta$. But in this case it must accept without moving to the blank symbol to the left side of the 1, which would cause an infinite sequences of left moves; so it must also accept all $x = 1^+$. In all cases $L(M) \setminus \{1\} \neq \emptyset$ leading to a contradiction. □

But, as showed in [2], there are one–state Turing machines that can recognize non–regular languages, i.e. some of them are more powerful than finite automata.

In the next sections we will prove that they are even more powerful.

3 A powerful one–state Turing machine

A one–state Turing machine has no internal memory, but the symbols on the tape cells can be used to *store* the following information:

- cell has been visited n mod k times (k fixed);
- head has left the cell going to the right;
- head has left the cell going to the left.

This information is limited, but it is enough to implement both an *unary counter* and a *binary counter*. Informally we define a Turing machine with this behavior:

- the input is divided in two parts: the left part (U^*) is an unary counter; whenever the head finds a U it marks it as *counted* writing C and bounces back to the right part;
- the right part (0^*) – initially filled with 0s – is a binary counter: whenever the head finds a zero 0 it transforms it to a 1 and bounces back to the left part; when it finds a 1 it marks it as a zero Z and continues towards the right to "propagate" the *carry*; when going back to the left part it will change Z to 0;
- at the (rightmost) end of the input there is the special symbol h for which no transition is defined and causes the machine to halt.

Formally, we define a one–state Turing machine $M_{cc} = \langle \{q\}, \Sigma, \Gamma, q, \{h\}, \delta \rangle$ over a three symbols input alphabet $\Sigma = \{u, 0, h\}$; the tape alphabet has eight symbols: $\Gamma = \{\sqcup, u, U, 0, 1, Z, C, B\}$; the complete transition table $\delta : \Gamma \to \Gamma \times \{L, R\}$ is:

Read	Write	Move	Description
u	U	R	Move towards the center and prepare the unary counter
0	1	L	Increment binary bit and go back left
U	C	R	Mark as counted an element of the unary counter
1	Z	R	Increment binary bit and propagate the carry
Z	0	L	Restore the zero and continue to the left
C	B	L	Skip C and continue to the left
B	C	R	Restore the Cs and continue to the right
$\sqcup$	$\sqcup$	L	Run left forever
h			Undefined (halt)

Figure 1 shows an example of a computation on input $uuuu00h$.

```
1   _ [u] u  u  u  0  0  h  _   : move to the center
2   _  U [u] u  u  0  0  h  _
3   _  U  U [u] u  0  0  h  _
4   _  U  U  U [u] 0  0  h  _   : bin counter is 00
5   _  U  U  U  U [0] 0  h  _
6   _  U  U  U [U] 1  0  h  _   : bin counter is 10
7   _  U  U  U  C [1] 0  h  _   : unary counter is C
8   _  U  U  U  C  Z [0] h  _
9   _  U  U  U  C [Z] 1  h  _
10  _  U  U  U [C] 0  1  h  _   : bin counter is 01
11  _  U  U [U] B  0  1  h  _
12  _  U  U  C [B] 0  1  h  _
13  _  U  U  C  C [0] 1  h  _   : unary counter is CC
14  _  U  U  C [C] 1  1  h  _   : bin counter is 11
15  _  U  U [C] B  1  1  h  _
16  _  U [U] B  B  1  1  h  _
17  _  U  C [B] B  1  1  h  _
18  _  U  C  C [B] 1  1  h  _
19  _  U  C  C  C [1] 1  h  _   : unary counter is CCC
20  _  U  C  C  C  Z [1] h  _
21  _  U  C  C  C  Z  Z [h] _   : halt
```

Figure 1: Computation on input $uuuu00h$ ("_" represents the blank symbol $\sqcup$)

We call L_{cc} the language defined by the inputs on which the Turing machine M_{cc} halts (i.e. reach the halting symbol h for which no transition is defined and doesn't run forever):

$$L_{cc} = \{x \mid M_{cc}(x) \text{ halts}\}$$

We don't need to characterize exactly the language L_{cc}; but if we restrict the input to strings of the form u^*0^*h, we can prove the following property:

Theorem 3.1. *If $x \in \{u^n 0^m h \mid n, m \geq 0\}$ then $x \in L_{cc}$ if and only if $n \geq 2^m - 1$.*

Proof. By construction the left part u^n behaves like an unary counter; while the right part 0^m behaves like a binary counter. For every increment of the binary counter the unary counter is also incremented, so the head can reach the rightmost h only if $n \geq 2^m - 1$; if $n < 2^m - 1$ then the head reaches the blank symbol on the left of the input and continues in that direction forever. □

We can say that, if the input is "well-formed", the one–state Turing machine can *compare* a number represented in unary with a number represented in binary.

Using a similar technique we could build a one–state Turing machine that implements an arbitrary k-ary counter or a one–state Turing machine that compare a k-ary number with a k'-ary number, i.e. it can compare two numbers written in different bases.

4 Computational complexity

We define the languages:

$$L_{u0h} = \{u^n 0^m h \mid n, m \geq 0\}$$

$$L'_{cc} = \{u^n 0^m h \mid n, m \geq 0 \wedge n \geq 2^m - 1\}$$

which, by Theorem 3.1, are related to each other and to L_{cc} in this way:

$$L'_{cc} \subseteq L_{u0h}$$
$$L'_{cc} \subseteq L_{cc} \cap L_{u0h}$$

We can prove that L_{cc} is not context–free (CF) using the well known property of context–free languages: if L is CF and R is regular then $L \cap R$ is CF.

The following is immediate:

Lemma 4.1. *$L_{u0h} = \{u^n 0^m h \mid n, m \geq 0\}$ is regular.*

and using the pumping lemma for context–free languages we prove the following:

Lemma 4.2. *L'_{cc} is not context–free.*

Proof. Suppose that L'_{cc} is CF. Let $p \geq 1$ be the pumping length. We pick the string

$$s = u^{2^p-1} 0^p h$$

which is in L'_{cc}. By the pumping lemma s can be written as $s = rvwxy$, with substrings r, v, w, x and y, such that *i)* $|vwx| \leq p$, *ii)* $|vx| \geq 1$, and *iii)* $rv^n wx^n y \in L'_{cc}$ for all $n \geq 0$. We can have the following cases:

- vwx is entirely contained in u^{2^p-1}; in this case we can pump zero times and we get the string $s' = u^k 0^p h$ with $k < 2^p - 1$, hence $s' \notin L'_{cc}$;
- vwx is entirely contained in 0^p; in this case we can pump one time and we get the string $s' = u^{2^p-1} 0^j h$ with $p < j$ and $2^{p-1} < 2^j - 1$, hence $s' \notin L'_{cc}$;
- vwx spans between u^{2^p-1} and 0^p; in the worst case $v = u^{p-1}$ and $x = 0^1$:

  $$u^{2^p-p}(u^{p-1})^n (0^1)^n 0^{p-1} \in L'_{cc}$$

  should hold for any $n \geq 0$, but for large enough n we have:

  $$2^p - p + n(p-1) < 2^{n+p-1}$$

  which puts the string out of the language. So we can conclude that L'_{cc} is not context–free.

□

We can finally conclude that the language recognized by the one–state Turing machine M_{cc} is not context–free:

Theorem 4.3. *L_{cc} is not context–free.*

Proof. Suppose that L_{cc} is CF. Then $L_{cc} \cap L_{u0h}$ is CF because L_{u0h} is regular (Lemma 4.1). But $L_{cc} \cap L_{u0h} = L'_{cc}$ which is not CF (Lemma 4.2) leading to a contradiction. □

If R is the class of regular languages, CF is the class of Context–Free languages, and TM_{1state} is the class of languages recognized by one–state Turing machines; by Theorem 2.4 and Theorem 4.3, we have :

Theorem 4.4.

$$R \not\subset TM_{1state}, \qquad TM_{1state} \not\subset CF$$

5 Conclusion

We proved that despite its limitations a one–state Turing machine can be used to implement a k-ary counter and "compare" two numbers written in different bases (the quotes are due to the non–standard definition of acceptance/reject of an input string). So the class of languages recognized by one–state Turing machines is neither a superset of the regular languages nor a subset of the context-free languages.

Acknowledgements

Thanks to Mikhail Rudoy for asking a related question [3] about single-tape 3-state Turing machines on cstheory.stackexchange.com: this paper is mainly a rework of the answer I gave to him.

References

[1] G. T. Herman. The halting problem of one state turing machines with n-dimensional tape. *Mathematical Logic Quarterly*, 14(7-12):185–191, 1968.

[2] G. T. Herman. The uniform halting problem for generalized one-state turing machines. *Information and Control*, 15(4):353 – 367, 1969.

[3] M. R. (https://cstheory.stackexchange.com/users/34401/mikhail rudoy). Class of languages recognizable by single-tape 3-state tms. Theoretical Computer Science Stack Exchange. URL:https://cstheory.stackexchange.com/q/38726 (version: 2017-08-01).

[4] T. Neary and D. Woods. Four small universal turing machines. *Fundam. Inform.*, 91(1):123–144, 2009.

[5] Y. Saouter. Halting Problem for One-State Turing Machines. Research Report RR-2577, INRIA, 1995.

[6] M. Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997.