# A Java visual Simulator of Turing Machines\*

Alessandro Burigana, Francesco De Martino, and Agostino Dovier

Dipartimento di Scienze Matematiche, Informatiche e Fisiche Università di Udine, 33100 Udine, Italy {burigana.alessandro, demartino.francesco}@spes.uniud.it agostino.dovier@uniud.it

**Abstract.** We present a graphical simulator of 1-Tape, k-Tapes, deterministic and non deterministic Turing machines. The simulator is written in Java and as such it runs on most platforms. During and after the computation it returns interesting features such as the amount of space visited/used and the number of steps. When simulating non-deterministic Turing machines it allows to browse the "tree" of non deterministic computations. It is developed purely for didactic purposes: it can be used in courses of Foundations of Computer Science, in courses of Computational Complexity, as well as in didactic projects with high schools.

**Keywords:** Turing Machines · Teaching Computer Science · Computability · Complexity.

# 1 Introduction

When computer science students meet computation models, and in particular Turing machines, they find useful (and even funny) testing their first programs using a simulator. Although encoding a simulator can be proposed as a useful exercise using a high level programming language, the availability of a graphical working simulator is in general very appreciated.

Several existing simulators are available, some of them running on Android and iOS. However, the availability of multi-platform simulators is rather limited, and, as far as the authors know, there is only one simulator for multi tape deterministic Turing machines, and no simulators at all for non-deterministic Turing machines. With this *communication* we would like to present a graphical simulator of 1-Tape, k-Tapes, deterministic and non deterministic Turing machines that, being written in Java, runs on most operating systems, and in particular it is tested on Windows, Linux, and macOS. The simulator (.jar) is available for free download and use from http://clp.dimi.uniud.it/sw/.

We rely on the syntax based on 5-tuples (q, s, q', s', L/R) as originally presented by Alan Turing in his seminal paper On Computable Numbers, with an Application to the Entscheidungsproblem [13]. The syntax is slightly extended for multi tape Turing machines as suggested in, e.g., [8, 4], while [10] and [1] used

<sup>\*</sup> Copyright © 2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 $\mathbf{2}$ 

a simplified syntax (in a single step the machine either write a new symbol or move, but not the two things together).

As anticipated above, Turing Machine simulators are available. A complete analysis is out of the scope of this paper. We just survey some of them focusing on those based on a 5-tuple syntax. Some of them are obsolete, such as [6], running on MS-DOS and based on a character-based interface. Uber Turing Machine [12] runs only on Windows systems and it is not free SW. Tursi [11] and Alan [7] are instead multi-platform simulators. A particular remark is due to *Turing Machine Simulator* [14] that, being a free app running on Android systems, including smartphones, it can be of interest of students. All the listed proposals simulate single-tape deterministic Turing machines only. There are then other multi-platform simulators based on other representations such as Tuatara [5] that allows also to experiment with formal languages, and JF1ap [9] that allows to simulate multi-tape Turing machines.

The tool we present (split in two components, A-machine and NA-machine, N standing for Non deterministic) is devised as a support for students of Computer Science or Mathematics, but it can be used for didactic projects with high schools (e.g., PLS project). However, the authors hope that it can be of interest of passionate scientific readers of introductory books such as *The Universal Computer* [2] or *The millennium problems* [3]. NA-machine can, in fact, help the reader in investigating the fascinating NP world.

## 2 Turing Machines

A Turing Machine is a 4-tuple  $\langle Q, \Sigma, P, q_0 \rangle$  where Q is a finite set of (mental) states,  $q_0 \in Q$  is the initial state,  $\Sigma$  is the alphabet of the machine, that should contain at least two symbols: one symbol for the "empty" cell (blank) "\_" and another symbol (tally). P is a set of 5-tuples of the form (q, s, q', s', m) where  $q, q' \in Q$ ,  $s, s' \in \Sigma$  and  $m \in \{L, R\}$  where L stands for *left* and R for *right* such that for each pair (q, s) there is at most one tuple beginning with q, s in P to guarantee determinism.

An instantaneous description (ID) is a 4-tuple  $\langle q, \ell, s, r \rangle$  where q is the current state,  $\ell sr$  denotes the "interesting" portion of the tape:  $\ell$  the string at the left of the head, s the symbol currently read, and r the string at the right of the head. P allows us to determine the deterministic relation successor ( $\vdash$ ) among IDs, and hence, the notion of computation. A machine halts if (and only if) the current ID is  $\langle q, \ell, s, r \rangle$  and no tuples starting with q, s is in the program P.

Some common variations are allowed in Turing Machines. For instance, the possibility of staying in the same position (that can be simulated by the execution of two consecutive instructions L and R) We refer to this feature as S (for "Stay"). In [8] P is assumed to define a function  $\delta : Q \times \Sigma \longrightarrow (Q \cup \{\text{yes}, \text{no}, \text{halt}\}) \times \Sigma \times \{L, S, R\}$ . Basically, when the machine halts it reaches a pseudo-state that acts as final state. This is of course allowed by the standard definition by using three states to be selectively reached at the stop time, and by adding no 5-tuples in P starting with them. Ending with yes, no, or halt can be

exploited as output signal. Those terminating always with yes and no are used for decision purposes.

It is well-known that a Turing machine with tape limited on one side (e.g., on the left) is sufficient to prove the equivalence between the set of Turing computable function and the set of (partial) recursive function (see, e.g., [4]), the Theorem that led to the Church-Turing thesis. Sometimes this can be made explicit by requiring that the tape begins with a symbol  $\triangleright$  which imposes some constraints: it cannot be cancelled and the head simply move R (and possibly change state) when reading it.

Following [8], k-tapes Turing machines are allowed. In this case the use of  $\triangleright$  is explicitly required in all tapes. Other restrictions are added. The first tape contains the input. Machines terminating with halt compute functions: in this case the result can be retrieved in the tape k. The program P is therefore described by a function  $\delta: Q \times (\Sigma)^k \longrightarrow (Q \cup \{\text{yes, no, halt}\}) \times (\Sigma \times \{L, S, R\})^k$ . I/O k-tapes Turing machines have the same structure; some further requirements are added on the behaviour relative to tape 1 and k. These machines are used for studying classes in space and for analyzing the functions used for reductions between problems.

Last, non determinism must be kept into account (leading to the famous open problem P vs NP). Non deterministic Turing machines are simply one-tape machines<sup>1</sup> with the unique (but crucial) difference that P is no longer a function: more than one tuple starting with a pair q, s are allowed. This leads to a definition of  $\vdash$  which is no longer deterministic. Starting from an initial  $ID_0 = \langle q_0, \varepsilon, \triangleright, x \rangle$ ,<sup>2</sup> the machine can run a set of non-deterministic computations. This set can be visualized as paths in a tree rooted by  $ID_0$  and we refer to it as the *tree of non deterministic computations* (briefly, ND tree).

### 3 The deterministic Turing Machine simulator

The first tool we present in this paper is the A-Machine; its name is inspired from [13] where Alan Turing said

If at each stage the motion of a machine [...] is completely determined by the configuration, we shall call the machine an "automatic machine" (or a-machine).

The simulator is written in Java, using the graphical library JavaFX. It is multiplatform and tested under Linux, Windows, and macOS systems.

The alphabet  $\Sigma$  contains two symbols by default: # and *blank* (\_). # is used as an ASCII version of  $\triangleright$  (it is not a technical problem, but we experimentally realized that its absence in the keyboards decreases the usability) and starts every tape. New symbols can be added by selecting the option in the upper right square (see Fig. 1). Each state has its unique name and an optional description.

<sup>&</sup>lt;sup>1</sup> Multi tape non-deterministic Turing machines are easy to be defined.

 $<sup>^{2}\</sup>varepsilon$  denotes, as usual, the empty string.

#### 4 A. Burigana, F. De Martino, A. Dovier

States can be added by selecting the option in the lower right square. A new machine is endowed with the initial state  $q_0$  and the "pseudo" states yes, no, and halt (see Section 2).



Fig. 1. A-machine: Initial screenshot.

Program instructions are added by clicking "+" below the left table. There is a text field where a description of the instruction can be added. Instructions can be later easily modified or cancelled. Some facilities to sort instructions, symbols, and states are available. Programs can be saved and retrieved.

The machine has one tape as default. New tapes can be can be added or erased. All tapes start with #. The input in the tape can be written/erased by clicking on the pencil in the right low corner.

Execution can be launched/interrupted by pressing on the common animation buttons. In Fig. 2 we report one example of execution. From the menu Computation (see Fig. 1), the user can select the Computation log and get all details (step by step) of the computation, the number of computation steps (info related to the time complexity), and the numbers of total cells and of new cells used (info related to space complexity).

The simulator have also other features that can be read in the manual available at the download site. We just give a quick overview here. I/O Turing machines can be generated and used by the tool. The programmer can choose if generate a new "standard" Turing machine or a I/O one. In the latter case the machine starts with 3 tapes. Verification of the constraints of I/O Turing machines are checked during programming. The interpreter is capable of using other machines as sub-routines (Turing referred to them as *skeleton tables*). They can be selected by specifying the path of the corresponding file. With this trick it is possible to define (mutually) recursive machines. The interpreter is capable of defining and running more machines at the same time.



**Fig. 2.** A-machine, after one simulation of a two-tapes deterministic Turing machine deciding in time O(3n) the language  $\{x \in \{0,1\}^* : x \text{ is palindrome}\}$ .

# 4 The non deterministic Turing Machine simulator

Although there is no proof of the fact that the main deterministic and non deterministic time computational classes are different (or equal) every computer scientist has experienced the exponential blow up that emerges in the simulation of a non deterministic Turing Machine with a deterministic one, when building and/or visiting the tree of non-deterministic computations. This intrinsic difficulty often leads students to avoiding the activity of programming non deterministic Turing Machine even for simple examples. The simulator we are presenting here could represent a support in the early stages of the study of non deterministic classes by automatically executing the code in a breadth first fashion and allowing the user to zoom on the interesting part of the ND tree. Handling of graphics is in this case demanded to the graphical Java environment Swing.

We refer as NA-machine (non deterministic A-machine) to this version of the simulator. The overall appearance of the program and user interface is the same as of A-machine for adding symbols, states, instructions. The constraint that for each pair q, s there is at most one instruction beginning with q, s is of course relaxed to allow non-determinism.

Three windows reporting computation data can be analyzed. *Statistics* keeps track of some general info on the ND tree and of the number of "yes" node that has been reached; *ND Tree* shows the main structure of the ND tree: the picture of the tree can be enlarged and, clicking on a node, the corresponding ID is shown; *Instantaneous description* reports the list of the all IDs associated with the nodes of the ND tree. See Fig. 3 for an example.

#### 6 A. Burigana, F. De Martino, A. Dovier



**Fig. 3.** *NA*-machine: observe the non determinism introduced by instructions 2 and 3. The ND tree can be reduced/enlarged (with usual "mouse/touchpad" techniques). Nodes representing halting with yes/no are green/red, resp. Clicking on a node, the corresponding state, tape content, and head position are reported. This is the output of the computation on input #00111 and the top gren node of the level 3 is selected.

# 5 Conclusions

We have presented a Java simulator for one tape, multi tape, deterministic and non deterministic Turing machines. The language of the interpreter (Italian vs English) can be selected modifying a settings file. The authors are ready to modify the simulators and/or add different features according to user requests.

# References

- 1. N. J. Cutland. An Introduction to Recursive Function Theory. CUP, 1980.
- 2. M. Davis. The Universal Computer: The Road from Leibniz to Turing. CRC, 2011.
- 3. Keith Devlin. The Millennium Problems. Granta Books, 2002.
- A. Dovier and R. Giacobazzi. Fondamenti dell'Informatica: Linguaggi Formali, Computabilità, Complessità. Bollati Boringhieri, 2020.
- 5. J. Foulds. Tuatara Turing Machine Simulator, University of Waikato. https://sourceforge.net/projects/tuataratmsim/
- J. Kennedy. Turing machine simulator. Santa Monica College, 1996. http://archives.math.utk.edu/software/msdos/miscellaneous/jkturing/
- 7. D. Neuber, O. Pahl, and D. Seichter. Alan. University of Applied Sciences Rosenheim. http://alan.sourceforge.net/
- 8. C. H. Papadimitriou. Computational Complexity. Addison-Wesley, 1994.
- 9. S. H. Rodger. JFLAP. Duke University. https://www2.cs.duke.edu/csed/jflap/
- H. J. Rogers. Theory of Recursive Functions and Effective Computability. The MIT Press, 1988.
- 11. C. Schätzle. Tursi. Univ of Freiburg. https://schaetzc.github.io/tursi/index.html
- 12. SuperUtils. Uber Turing Machine. https://uber-turing-machine.soft32.com/

- 13. A. M. Turing. On computable numbers with an application to the entscheidungsproblem. *Proc. of the London Math. Society*, 42(2):230–265, 1936.
- 14. I. Yudov. Turing Machine Simulator. https://play.google.com/store/apps/details?id=ru.ilyayudov.TM.

Research partially supported by the University of Udine PRID ENCASE project.