A journey of a thousand miles begins with a single step

## Contents

1. **Introduction** \( \rightarrow 3 \)

2. **Multi-tape Turing Machine** \( \rightarrow 3 \)
   2.1 Definition \( \rightarrow 3 \)
   2.2 Specifications \( \rightarrow 4 \)
   2.3 Sets \( \rightarrow 5 \)
   2.4 Codification \( \rightarrow 6 \)

3. **Formal Languages** \( \rightarrow 9 \)
   3.1 Recursive-Enumerable Languages \( \rightarrow 9 \)
   3.2 Context-Free Languages \( \rightarrow 9 \)
   3.3 Regular Languages \( \rightarrow 10 \)

4. **Universal Turing Machine** \( \rightarrow 12 \)
   4.1 Definition \( \rightarrow 12 \)
   4.2 Usage \( \rightarrow 13 \)
   4.3 Options \( \rightarrow 13 \)
   4.4 Examples \( \rightarrow 16 \)

5. **Algorithms** \( \rightarrow 16 \)
   5.1 FTM \( \rightarrow 17 \)
   5.2 ETM \( \rightarrow 19 \)
   5.3 DTM \( \rightarrow 20 \)
   5.4 NTM \( \rightarrow 23 \)
   5.5 Another DTM \( \rightarrow 25 \)
   5.6 Recursive Language \( \rightarrow 25 \)
   5.7 Context-Free Language \( \rightarrow 27 \)
   5.8 Regular Language \( \rightarrow 27 \)

6. **Version** \( \rightarrow 28 \)
   6.1 Implementation \( \rightarrow 28 \)
   6.2 License \( \rightarrow 29 \)
1 Introduction

The present document has been written only for educational purposes. The objective is to help to understand, design, and test the concept of the automata (such as DFA, PDA, and TM) using a Universal Turing Machine (UTM). Therefore, this is not a complete documentation but goal-oriented according our framework —do not worry... it is not a Turing-Test!... maybe. At the same time, it is the manual for our particular implementation of the UTM: utm.

The TM has been invented in 1936 by Alan Turing as a model of computation. At date, the power —no understand speed— of this model has not been improved (it is a simulation of any hand computation). For a detailed information, you can read the theoretical paper On Computable Numbers, With an Application to the Entscheidungsproblem: A correction (Alan Turing, 1937) or, in a more accessible language, the educational book Introduction to the Theory of Computation (Michael Sipser, 2006), or, in a more general perspective, The Emperor’s New Mind (Roger Penrose, 1989).

The main goal is to parse/compute algorithms written for a Multi-tape Turing Machine working in a functional way (FTM). Algorithms for Enumerator Turing Machines are also accepted (ETMs). Moreover, as a particular cases, Finite Automata —with and without stack— and 1-tape Decider Turing Machines (DTMs) are also included (following Sipser’s book). This last type corresponds to the classical TM.

In order to test immediately the utm with finite automata or TMs, you can go directly to read Sec. 6.1 and 4.4 —nevertheless, a lecture of all document is recommended.

2 Multi-tape Turing Machine

2.1 Definition

Let $M_k$ be a $k$-tape MTM: $M_k = (\Sigma, \Gamma, Q, \delta, q, F)$, where

- $k$ is the number of tapes (each one has an infinite number of cells);
- $\Sigma$ is the input alphabet;
- $\Gamma$ is the tape-alphabet;
- $Q$ is the set of states;
- $\delta$ is the set of transitions;
- $q$ is the initial state; and
- $F$ is the set of final states.

The transitions set is defined by the function

$$\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^{k-1} \times \Omega^k.$$
2.2 Specifications

where \( \delta(q_i, \alpha) = (q_j, \beta, \omega) \), and \( q_i \in Q, \alpha \in \Gamma^k, q_j \in Q, \beta \in \Gamma^{k-1}, \Omega \) is the set of head movements \{left, right\}, and \( \omega \in \Omega^k \). At the first time, \( M_k \) is in the initial state \( q \) with an input word in the first tape. When the machine is in state \( q_i \) and the sequence of symbols read by their heads \( h_1 \cdots h_k = \alpha \), the next state becomes \( q_j \), each \( h_i \in \{2, \ldots, k\} \) writes the symbol \( \beta_i \), and each \( h_i \in \{1, \ldots, k\} \) moves according \( \omega_i \). Note that the first tape has only read-head (usually, no write-head for the input is useful). If none transition is possible, the automaton halts.

A special situation appears when a tape-head is at the first cell and a left movement is necessary. Some designs work leaving the head at the first cell (as in Sipser’s book case) and others consider this situation an error. Our implementation use this last option because the user always takes in consideration this special case.

2.2 Specifications

Basic questions are:

- \( M_k \) works in a functional way (i.e., \( F = \emptyset \)) and the result is written on an specific output-tape. A decisional behavior on any set is achieved computing its characteristic function \( \chi \).

- We have \( k = in + work \) tapes, where \( in = 1 \) is the input-tape (only read, tape \#1) and \( work > 0 \) are the work-tapes (read/write, tapes \#2..\#k). Thus, \( k > 1 \).

- When \( M_k \) halts, the result is in tape \#k (output-tape) starting at the first cell.

- Without loss of generality, and in order to simplify the designs, we consider a no-move movement for the tape-heads.

- The \( \mathcal{N} \) set can be represented in any base (we consider \( 0 \in \mathcal{N} \)) but the designer needs to consider and comment it inside the \( M_k \) design and documentation, respectively (see Sec. 2.4).

- If the precondition of the input is not fulfilled (i.e., the input does not belong to \( \Sigma^* \) according the specific format noted in the public documentation), any \( M_k \) behavior is possible —without any condition.

Note that other definitions of TMs are possible —without increasing the computational power— including other kinds of machines —as cellular automata. Some examples could be: TMs with infinite tapes in both directions, the Wolfram TM, the Langton’s Ant, the Conway’s Game of Life, the \( \lambda \)-calcul, etc. In any case, the power is always the same (i.e., Church-Turing Thesis).

In our implementation, you can consider that the UTM deals with a four particular TM models:
2.3 Sets

**Functional** (FTM) Multi-tape Functional TM for to compute functions according the previous definition (see Sec. 2). This is the default case. The minimum number of tapes is two (input-tape and output-tape):

\[ \delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^{k-1} \times \Omega^k, \ k > 1 \]

**Enumerator** (ETM) Enumerator TM for to print an enumeration of a set (formal language). No input is necessary (i.e., the input-tape is not used) but it needs, as a minimum, one work-tape and the output-tape (i.e., as a printer device):

\[ \delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \Omega^k, \ k > 1 \]

**Decider** (DTM) Decider TM for to compute characteristic function of a set (formal language). The halting can be achieved only by two special states: reject and accept. The number of tapes is exactly one (i.e., only the output-tape is used whereas the input-tape contains the original input—duplicated in the output-tape):

\[ \delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \Omega \]

**Non-deterministic** (NTM) Non-deterministic TM as a general case of the previous models because non-deterministic behavior is possible in any of them (see option -n in Sec. 4.3). Usually, in our context, the distinction between the functional (NTM-F), enumerator (NTM-E), and decider (NTM-D) models is not necessary because we consider the non-deterministic transition

\[ \delta(q_i, \alpha) = \{(q_{j_1}, \beta_1, \omega_1), \ldots, (q_{j_n}, \beta_n, \omega_n)\} \]

as the sequence:

\[ \delta(q_i, \alpha) = (q_{j_1}, \beta_1, \omega_1) \]

\[ \ldots \]

\[ \delta(q_i, \alpha) = (q_{j_n}, \beta_n, \omega_n). \]

This agreement is valid for any kind of codification in this document.

More details about these models appear in Sec. 2.4.

### 2.3 Sets

With respect the sets involved:

- The universal set is

\[ \text{ALPHABET} = \{a_{32}, a_{33}, \ldots, a_{125}, a_{126}\} = \{\$, !, \ldots, \#, \sim\},\]

where \(a_i\) is the \(i\)-character in the American Standard Code for Information Interchange (ASCII 7 bits).
2.4 Codification

- The symbol $a_{32} = \sqcup$ (space symbol) will represent a free-cell in a tape.
- The set
  \[
  TRANSITION = \{a_{42}, a_{64}, a_{91}, a_{93}, a_{124}\} = \{\ast, @, [, ], |\}
  \]
  is a reserved set used in the codification of any $M_k$ (see Sec. 2.4).
- The $\Sigma$ alphabet is a subset in $ALPHABET - TRANSITION - \{\sqcup\}$.
- The $\Gamma$ tape-alphabet fulfills
  \[
  \Gamma \subseteq ALPHABET - TRANSITION \land \Sigma \subset \Gamma \land \sqcup \in \Gamma.
  \]
- The set of states is $Q = \{q_1, \ldots, q_n\}$, where
  \[
  q_i \in (ALPHABET - TRANSITION - \{\sqcup\})^+ \text{ as for example forward-looking_for_3_consecutive_free_cells.}
  \]
- The set of boolean values is $B = \{a_{48}, a_{49}\} = \{0, 1\} = \{false, true\}$. However, you can decide any kind of representation for $B$ if you note it in the $M_k$ public documentation (see Sec. 2.4).

2.4 Codification

We need to choose an arbitrary codification system for the set of our TMs. An $M_k$ will be coded in a pure standard ASCII file with extension $tm$ containing the basic information: alphabets ($\Sigma$ and $\Gamma$) and transitions ($\delta$). The set of transitions are enumerated in any order representing all the transitions that appear in the edges of a TM-diagram —the only exception will be the first coded transition that it have to have its origin in the initial state. The rules to follow are:

- The first line has $\Sigma$: $\Sigma = [s_1 \cdots s_n]$, where each symbol $s_i \in \Sigma$ (the next information in the line will be ignored: private documentation).
- The second line defines the new alphabet
  \[
  \Delta = \Gamma - \Sigma - \{\sqcup\},
  \]
  codified as $\Delta = [s_1 \cdots s_m]$ (the next information in the line is also private documentation). Note that $\Sigma$ never is empty whereas $\Delta$ could be ($\Delta = []$).
- All the next information in the $tm$-file can be classified in three kinds identified by the first character of each line:
1. Public information.
   The minimum data about $M_k$ is: description (informal), functional
   definition (formal), input format (with example), output format (with
   example), design date, version number, UTM version number (of the
   UTM codification system —see Sec. 6.1), author names, and a con-
   tact way (e.g., e-mail). For this data, use respectively the keywords
   Description, Function, Input, Output, Date, Version, UTM-version,
   Author, and Contact inside the lines.

2. Transition.
   $\delta(q_i, \alpha) = (q_j, \beta, \omega)$, where \{q_i, q_j\} \subseteq Q, \alpha \in \Gamma^k, \beta \in \Gamma^{k-1}, \text{and}
   \omega \in \Omega^k$ is coded as $[q_i|\alpha] = [q_j|\beta|\omega]$ —note that $\beta \in \Gamma^k$ in ETMs
   and $k = 1$ in DTM$. All the information of the line following the
   transition is considered private documentation.

3. ALPHABET $\{ |, [ \}$ Private documentation.
   Any comment that the designer considers.

   • In order to code one transition, three considerations:

1. The joker symbol ($a_{42} = *$)
   It plays three roles inside a transition:
   - **Reading** The symbol $\alpha_i$ read is not used (i.e., ignored).
   - **Writing** The symbol $\beta_i$ to write is the same symbol read (i.e., $\beta_i = \alpha_i$ or, equivalently, it is ignored for the write-operation).
   - **Head moving** The head is ignored (i.e., $\omega_i = *$ implies a no-move
     for $h_i$). The no-move “movement” (the transition says that the
     head need to be in the same cell) and joker symbol (the tran-
     sition ignores this tape-head) are semantically and syntactically
     different but functionally identical.

2. The complementary symbol ($a_{61} = @$)
   It can be used only inside the words $\alpha$ and $\beta$ represenitng the
   complementary set with respect to the set of symbols that appears in
   the same state and tape. The specific universal set is $\Gamma$ except for
   the input-tape where it is $\Sigma$. For each symbol represented in $\alpha$ by
   $\emptyset$, the same symbol is also represented by the first occurrence of $@$
   in $\beta$, if it exists. Therefore, the number of these symbols in $\alpha$ has
   to be greater or equal than in $\beta$. Note that the use of this symbol
   together with the joker symbol in the same state and tape has no
   sense. Therefore, we declare this kind of use incompatible. In order
   to accelerate the computation, this symbol is preprocessed —you can
   see its application showing the $\delta$ set (see -3 option in Sec. 4.3).

3. The head-movements ($\Omega$)
   The codification for the final set of tape-head movements is
   $$\Omega = \{ \text{ignored, left, no-move, right} \} = \{a_{42}, a_{60}, a_{61}, a_{62}\} = \{*, <, =, >\}$$
where *ignored* implies *no-move*.

Be careful with the use of the special symbols. They exist for to do more friendly the \( \delta \) designs, however they can easily produce unexpected behaviors or non-deterministic designs if you do not use it accurately.

Example of a transition codification for \( k = 5 \):

\[
\delta(q_{17}, 1****) = (q_{23}, * \sqcup 1*, =< * > *)
\]

is coded by

\[
[q_{17}|1***|] = [q_{23}|*\sqcup1*| =< * > *].
\]

And for \( k = 8 \), you could find in the \( tm\)-file:

\[
[forward_\text{for}_\text{free}|1B@**A21] = [backward_\text{for}_\text{free}|3_1\sqcup2| >****<].
\]

The last notes:

- The number of transition cannot be null (i.e., \(|\delta| > 0\)).
- The initial state is defined as the first state that appears in the first transition (i.e., \( q \) is the first component of \( \delta_1 \)).
- The relation of lengths between \( \alpha \) and \( \beta \) in the first transition \( \delta_1 \) defines the model used in the design (FTM, ETM, or DTM).
- In DTM designs:
  - The predefined states *reject* and *accept* cannot appear in the domain of \( \delta \).
  - The previous states can only appear in the range of \( \delta \).
  - If a halt situation appears in a different state, the *reject* consideration is applied.
- In NTM designs note that a \( \epsilon \)-transition (or \( \lambda \)-transition in other definitions) \( \delta(q, \epsilon) = \{q_1, \ldots, q_n\} \) corresponds to the following codifications (\( i \in \{1, \ldots, n\} \) and \( k > 1 \)):

  \[
  \begin{align*}
  \text{FTM} & \quad [q|*^k] = [q_i|*^{k-1}|*^k] \\
  \text{ETM} & \quad [q|*^k] = [q_i|*^k|*^k] \\
  \text{DTM} & \quad [q|*] = [q_i|*|*]
  \end{align*}
  \]

A set of TM examples are shown in the appendix (one for each model). They solve the same language in order to note the differences in their respective algorithms.
3 Formal Languages

In this section, we recall three classes of formal languages directly related with automata and grammars (see any book about formal languages and automata for more details).

3.1 Recursive-Enumerable Languages

If all the words of a formal language \( L \) are accepted by a TM (i.e., in any positive boolean way for a FTM —its characteristic function— or accepted by a DTM), we can say that \( L \) is a recursive-enumerable or Turing-recognizable language. A recursive-enumerable set can be enumerated by a TM (ETM case) maybe without order and with repetitions. The set of recursive-enumerable languages corresponds to the unrestricted grammars in the Chomsky hierarchy.

Moreover, if we can also reject, in the same way, all the words outside \( L \), then it is a Turing-decidable or recursive language. A recursive set can also be enumerated by a TM but with order and without repetitions. Note that any recursive languages is also recursive-enumerable, and the reverse case it is not true as a general statement.

A simple example of recursive —and at the same time, recursive-enumerable— set is shown in Sec. 5.6.

3.2 Context-Free Languages

The set of decidable languages by a non-deterministic pushdown automaton (PDA) constitutes the class of context-free languages. It is a proper subset of the recursive languages and is equivalent to the context-free grammars in the Chomsky hierarchy.

We can test any PDA because it works as an specific case of TM (one work-tape used only as a stack). We convert each PDA-design in a FTM that computes the characteristic function of the PDA language (a pre-parser is used). The result of the conversion is saved in a \( tm \)-file that is actually the TM executed. This file is a proof of the inclusion of this context-free language in the recursive set. The FTM that simulates the PDA works in a general non-deterministic way and reject/accept the input —remember that a deterministic PDA is a particular case —and a proper subset of the non-deterministic class.

In order to made a PDA-design, it is necessary to follow the next specific rules:

- The white symbol \( ⊥ \) cannot be used in \( Σ \) because, in the current implementation, it represents the empty stack —this fact is transparent to the designer but it can be observed in any debug situation.
- \( Γ \) is now the stack alphabet. As occurs in \( Σ \), the \( ⊥ \) symbol cannot be used.
- A set \( F \) of final states is defined.
3.3 Regular Languages

The transitions are defined as \( \delta : Q \times \Sigma \times \Gamma_\epsilon \to P(Q \times \Gamma_\epsilon) \), where a transition \( \delta(q_1, a, b) = (q_2, c) \) means that in state \( q_1 \), reading the input symbol \( a \), and with the symbol \( b \) on the top of the stack, we move to state \( q_2 \), pop the stack, and push the symbol \( c \) in it.

- The \( \epsilon \) can be used in any place for \( a, b, \) and \( c \) representing that any input symbol is read, any pop is done, and any push is done, respectively. Then, the case \( \delta(q_1, \epsilon, \epsilon) = (q_2, \epsilon) \) is a pure \( \epsilon \)-move.

- The input is only accepted if it has been fully read and the automaton is in a final state (the stack can be empty or not).

The codification of this model follow the same TM general rules (see Sec. 2.4 but taken into consideration:

- \( \Sigma \) is coded in the first line in the same way than in TMs.

- \( \Gamma \) is codified as \( \Delta \) definition in the second line of the TM-codifications but replacing the keyword Delta by Stack.

- A third line with the definition of \( F = \{q_1, \ldots, q_n\} \) appears:

\[
\text{Final} = [q_1] \cdots [q_n].
\]

- The non-determinism in \( \delta \) is expressed option by option (see Sec. 2.2).

- A transition \( \delta(q_1, a, b) = (q_2, c) \) is coded as \( [q_1|ab] = [q_2|c] \).

- \( \epsilon \) is represented by a single dot (.)

According to the previous comments, when the \texttt{utm} loads a PDA, two new files are created: a \texttt{.def} file containing the definition of the PDA loaded, and a \texttt{tm-file.pda.tm} containing the TM that simulates the PDA — proof by construction that the language of this PDA is Turing-decidable. For all PDA files, the flags \texttt{−n} and \texttt{−B} are enabled (see Sec. 4.3). A simple PDA example is shown in Sec. 5.7.

### 3.3 Regular Languages

The set of decidable languages by a non-deterministic finite automaton (NFA) constitutes the class of regular languages. It is a proper subset of the context-free languages and is equivalent to the regular grammars in the Chomsky hierarchy and to the regular expressions.

We can also test any NFA because it works as an specific case of PDA (without stack) and TM (without tapes). We convert each NFA-design in a FTM that computes the characteristic function of the NFA language (a pre-parser is used). The result of the conversion is saved in a \texttt{tm-file} that is actually the TM executed. This file is a proof of the inclusion of this regular language in the recursive set. The FTM that simulates the NFA works in a general non-deterministic way and reject/accept the input. Remember that a deterministic
finite automaton (DFA) is a NFA particular case and that any NFA can be converted to a DFA. Thus, the class of languages produced by the deterministic and non-deterministic models are the same.

In order to make a NFA-design, it is necessary to follow the next specific rules:

- The symbol ⊥ cannot be used in Σ (as in PDA case).
- The Γ alphabet disappears.
- A set $F$ of final states is defined.
- The transitions are defined as $\delta : Q \times \Sigma_\epsilon \rightarrow P(Q)$, where a transition $\delta(q_1, a) = q_2$ means that in state $q_1$ and reading the input symbol $a$, we move to state $q_2$.
- The $\epsilon$ can be used in the place of $a$ representing that any input symbol is necessary for this movement (i.e., an $\epsilon$-move).
- The input is only accepted if it has been fully read and the automaton is in a final state.

The codification of this model follow the same PDA and TM general rules (see Sec. 2.4) but taken into consideration:

- $\Sigma$ is coded in the first line in the same way than in TMs.
- $\Gamma$ is replaced in the second line by the definition of $F = \{q_1, \ldots, q_n\}$: $Final = [q_1|\ldots|q_n]$ (as in PDA).
- The non-determinism in $\delta$ is expressed option by option (see Sec. 2.2).
- A transition $\delta(q_1, a) = q_2$ is coded as $[q_1|a] = [q_2]$.
- The $\epsilon$ is represented by a single dot (.) as in PDA case.

Analogously to PDA files and the previous comments, when the utm loads a NFA, two new files are created: a $nfa.def$ file containing the definition of the NFA loaded, and a $tm$-file ($nfa.tm$) containing the TM that simulates the NFA —also it is a proof by construction that the language of this NFA is Turing-decidable. However, if the original file is deterministic, the $dfa$ extension replaces the $nfa$ one in order to note this case. If the user decides to work with the Minimal Finite Automaton (MFA, flag $-m$ in Sec. 4.3), we have a new automaton and the files $mfa.def$ and $mfa.tm$ appear containing their definition and TM-codification, respectively. Moreover, if the original file was not deterministic, the $dfa.def$ file corresponding to the original NFA file is also saved (note that it can be different of a user input DFA containing inaccessible states). For all NFA files, the flag $-n$ is only enabled for non-deterministic cases (i.e., disabled for $dfa.tm$ and $mda.tm$ TMs), and the flag $-B$ is always enabled (see Sec. 4.3). A simple NFA example is shown in Sec. 5.8.
4 Universal Turing Machine

4.1 Definition

The UTM simulates the computation of any TM $M_k$ on any input:

$$UTM(number, input),$$

where the $number$ represents a code of any TM. As a particular cases, the UTM works also with $tm$-files corresponding to PDA and NFA designs — coded according the rules of Sec. 3.2 and Sec. 3.3, respectively. In our case, the UTM implementation $utm$ works according the following:

- Simulation of MTM, ETM, DTM, and NTM (see Sec. 2).
- The $number$ is represented by the ASCII text file-name (i.e., its path with or without $tm$ extension) corresponding to a design of an $M_k$ according the previous rules (see Sec. 2.4).
- The $input$:
  - Will be asked on-line or, optionally, the last parameter in the command line (this use is called silent mode).
  - If the $input$ is not written like the function definition (see option -1 in Sec. 4.3), the output can be anything (i.e., undefined, wrong, or the expected result — you are feeling lucky!).
  - A null $input$ represents $\epsilon$ (i.e., null-word). It can be inputted in a standard way (i.e., only <Enter>) or explicitly (i.e., the text word epsilon).
  - The space symbols are ignored and also mark the end of the $input$.
  - If the $M_k$ is an ETM, the $input$ is omitted.
- The $output$:
  - If and only if a FTM halts, the content of the output-tape is shown as the definitive solution.
  - In an ETM, every time that the head writes a $\sqcup$ in the output-tape (interpreted as a separator elements in the sequence), this tape is shown (with a previous indication of the transition number).
  - For DTMs, the result is not in the output tape because only the halt states (i.e., accept or reject) are taken into account and they are shown as a $output$. 

Fundamentals of Computing 12 Universal Turing Machine
4.2 Usage

The command line call is

\[ \text{utm \ [-\{hrv\} \ | \ (-\{01234bBcdDmnptuxy\}\})* \ function \ [input]],} \]

where function is the path of a tm-file (see Sec. 2.4) and input is the word to compute. It is important to note that if the input appears in the command line, the utm will work in silent mode (i.e., any dialogue occurs). Therefore, this is a usefully mode for batch process. In silent mode, some kind of input would have to be quoted in order to avoid the shell interpreter (e.g., “(11,101)”). Next, the previous options with some usage examples are commented.

4.3 Options

The current set of options are:

-0 Output of information: none information is shown (default).

-1 Output of information: the public documentation is shown before the computation.

-2 Output of information: the quantitative data of the machine is shown before the computation.

-3 Output of information: the set of transitions (\(\delta\)) is shown before the computation.

-4 Output of information: the configurations of the machine are shown.

-b Boolean generic flag: map a boolean output-symbol in \{false, true\}. If the output is in the set \{0, f, F, n, N\}, the value is false, and if it is in \{1, t, T, y, Y\}, the value is true.

-B Boolean specific flag: map a boolean output-symbols in \{0, 1\} to \{accept, reject\}, respectively. Automatically enabled in NFA/DFA/MFA and PDA tm-files and it always has priority over the -b flag.

-c Code flag: the user can input a finite subset \(N'\) of natural numbers \(N\) as \(N_{10}\) (decimal base) and they are code to their respective binary numbers before to be put in the input-tape. The process is reversed before to show the output-tape. The \(N'\) set is \{0, \ldots, 18446744073709551615\} and, for bigger values, you cannot used this option and you have to input the values in an standard way (i.e., \(N_2\) — binary base). This flag is a friendly option for human users in binary computations.

-d Debug flag: the debugging mode is enabled. All the configurations will be shown and, after a predetermined time-quantum (default is 1), the utm will ask you about the next time-quantum. The pseudo-debug flag is disabled (see -D option). There are two modes for debugging:
Raw mode It is the standard mode and the best option because the configurations are shown in a sequential way and you can see all the previous configurations and select any quantum. The answer to time-quantum question has the following options (< Enter > mode):

- **n > 0** New time-quantum (i.e., number of transitions to compute).
- **0** Continuous computation (i.e., no more time-quantum).
- **quit** Abort the current computation.
- **Any other answer** Continuous computation (i.e., it will be interpreted as a zero).

Unraw mode (see -u option). The configurations are overwritten and you cannot see the previous configurations but it could also help you looking for the bug. The time-quantum question are answered by one-keyboard mode (mainly by calculator keyboard):

- **Space** Time-quantum is one.
- **Enter** Time-quantum is one.
- **0** Continuous computation.
- **1..9** New time-quantum.
- **+** Up one the current time-quantum.
- **-** Down one the current time-quantum.
- ***** Twice the current time-quantum.
- **/** Half the current time-quantum.
- **q** Abort the computation.

-D Pseudo-debug flag: the pseudo-debugging mode is enabled. All the configurations will be shown with a delay of one second (trick: one second for each flag presence). The flag also works in unraw mode (see -u option) but is disabled by the more priority -d option.

-h Help flag: shows a brief summary of this section.

-m MFA flag: runs a Minimal Finite Automaton from an original NFA file. Additionally, it saves the definition of the original NFA (or DFA) determinized (.dfa.def) and its minimal automaton (.mfa.def). See Sec. 3.3 for more details.

-n Non-deterministic flag: as by default utm works in a deterministic way, you need to enable this flag if you have a non-deterministic design, otherwise it will stop the computation if a non-deterministic situation appears. This behavior is also a security control for your design. The non-deterministic situation is only checked in running-time. This flag is automatically enabled in NFA/PDA tm-files.

Except for the ETMs, if the word is accepted, a proof based on the path is saved (.tm.log). The path shows each transition followed by the #selection (random)/number of options.
4.3 Options

-\texttt{p} Parser flag: enable this flag if you only want to parse a \texttt{tm}-file. All the other options are not used if this flag is present.

-\texttt{r} Read-me flag: this documentation is shown.

-\texttt{t} Time flag: the computation-time is shown (it is the spent time between after the input and the last transition). The human time, TM time (transitions), and the speed (microseconds/transition) are presented. It is not sense neither with -\texttt{d} flag nor with ETMs but, in the other cases, it is useful in order to do time analysis. In the silent special mode, the flag put only the TM time after the output thinking in a possible batch computation making a set of data for future analysis.

-\texttt{u} Unraw mode: the tapes are shown at each configuration and updated in a overwritten way. It is an alternative to standard raw mode and it is possible to combine with the -\texttt{d} option.

-\texttt{v} Version number: critical number for us and designers in order to know the history and codification style in \texttt{tm}-files (i.e., \textit{UTM-version} in public comments —see Sec. 2.4).

-\texttt{x} Conjecture mode: you give permission to \texttt{utm} for conjecture undefined values (maybe it has good luck!). Of course, it is disabled for ETMs.

-\texttt{y} Dictionary flag: a micro-set of predefined UTM-words are translated after the input and before the output. This flag is usually used for test designs with numeric or alphabetic inputs. In this current version, we have:

\begin{verbatim}
NATURAL 4294967295
PRIME   4294967291
BINARY  (10)^50 = 1010101010\cdots1010101010
HELLO   KSXMYDHNIEB
ROMMEL  RUQ,LRJBCQ\cdots,QHXPMOBZB
CHURCHILL1 YQKOOUYMR\cdots,WXUDAOIHHU
CHURCHILL2 TSPIADJOEQ\cdots,TRHEBSVJS,
CHURCHILL3 TUYTIJUEO\cdots,VRXC,KEXJ
SHAKESPEARE YDGUBTZOPQ\cdots,HOQ,ZDHCMP
LAO-TZU MDMV,FMLJB\cdots,EDKQYDTUTI
SAGAN   USKGTOK,VK\cdots,OHZO,YDEOX
\end{verbatim}

The words from \textit{HELLO} to \textit{SAGAN} are famous sentences coded with the MTM design \textit{enigma} (see \url{http://en.wikipedia.org/wiki/Enigma_machine}). You can find it in the \textit{tm}-files examples set (see \textit{TM directory}) and you can run it with this input values in order to discover the complete sentence decoded (e.g., \texttt{utm -y -u -t TM/enigma SAGAN}).
4.4 Examples

List of some examples of calls to UTM:

- **utm function** Basic call for to do a computation by the TM coded in function.tm.
- **utm function input** Basic call for to do a computation of input by the TM coded in function.tm using silent mode (i.e., without any dialogue). It is useful in batch process or testing designs.
- **utm -p function** The file function.tm is only parsed.
- **utm -1 function** The file function.tm runs after show you the public information. This is the standard way of use.
- **utm -1 -d function** Like the previous example but debugging.
- **utm -1 -2 -3 -4 -t function** Before the computation, all the information about the TM is shown. Moreover, the configurations are shown after each transition. And after the computation, the time is also shown.
- **utm -c -t function** The computation is in base 2 and the user works in decimal base. The computation-time will be shown when the machine halts.
- **utm -b -d -n -y function** Running a non-deterministic design in debug mode. The input and output use the dictionary and output is mapped to boolean set.
- **utm -u -t function** Running in unraw mode and showing the total time. This is a cool way to compute.
- **utm automaton.pda** Running a PDA in non-deterministic way (flags -B and -n are automatically enabled by PDA). The files automaton.pda.def and automaton.pda.tm are created.
- **utm -m automaton.nfa** Running the MFA of the NFA file (automatically, flag -B is enabled and flag -n is only enabled for non-deterministic cases). The files automaton.nfa.def (only if the source file is not a DFA), automaton.dfa.def, automaton.mfa.def, and automaton.mfa.tm are created.

5 Algorithms

Next, using different algorithms (i.e., automata designs), an example of each sub-model of TM is shown for

\[
L = \{ww \in \{0,1\}^* \mid w \in \mathbb{N}_2\} = \{00, 11, 1010, 1111, 100100, 101101, \ldots\}.
\]

An example for \(\Sigma = \{a,b,c\}\) is also shown. Moreover, simple examples for recursive, context-free, and regular languages are included. A good way that
helps to understand any design is running it slowly by hand or automatically (e.g., `utm -D -u TM/n_2.n_2_f.tm`—see Sec. 4.3).

You can find their respective `tm`-files in the `TM` sub-directory (inside the `UTM` main directory). Other examples can be found in the same path.

5.1 FTM

In Fig. 1, a diagram of a FTM is depicted (`n_2.n_2_f.tm`).

```
Sigma=[01]
Delta=[$2]
```

Example using a FTM
| Name: concatenation of two binary numbers |
| Sigma = {0, 1} |

Figure 1: Diagram for the `n_2.n_2_f.tm` automaton.
5.1 FTM

\[ L = n_2n_2 \ (n_2 \ is \ a \ binary \ number) \]

Description: characteristic function of \( L \)

Function: \( f : \Sigma^* \rightarrow \text{Boolean} \)

\[ f(w) = 1, \ \text{if} \ w \ \text{in} \ L \]

\[ = 0, \ \text{otherwise} \]

Input: word \( w \) (e.g., 10111011)

Output: 0 or 1

Date: October 21, 2011

Version: 1.0

UTM-version: 0.2357111317192329313741

Author: Copyright 2011 Jaume Rigau

Contact: jaume.rigau@udg.edu

---

Is it epsilon?

\[
[\text{start}| ***] = [\text{halt}| ***0|===] \\
\]

-----------------------------

Is it 00? (minimum word)

\[
[\text{start}| 0***] = [0| ***0|>***] \\
[0|0***] = [00| ***] > *** \\
[00| ***] = [\text{halt}| ***1|===] \\
\]

-----------------------------

It can only start with 1

Where is the half of the word?

\[
[\text{start}|1***] = [\text{count}| $$$0|=>=>] \\
[\text{count}|0 **] = [\text{count}|01**|>**] \\
[\text{count}|1 **] = [\text{count}|11**|>**] \\
[\text{count}|0 1**] = [\text{count}|02**|>**] \\
[\text{count}|1 1**] = [\text{count}|12**|>**] \\
[\text{count}| **] = [\text{half}| ***|<<<<] \\
\]

-----------------------------

Going to half

\[
[\text{half}| **2**] = [\text{half}| *|<<<<] \\
[\text{half}| ***$] = [\text{left}| * |<<<<] \\
\]

-----------------------------

Going to first position

\[
[\text{left}|0***] = [\text{left}| ***|<<<<] \\
[\text{left}|1***] = [\text{left}| ***|<<<<] \\
[\text{left}|$$**] = [\text{match}| **|>***] \\
\]

-----------------------------

Matching

\[
[\text{match}|00**] = [\text{match}| **|>**] \\
[\text{match}|11**] = [\text{match}| **|>**] \\
[\text{match}| **] = [\text{halt}| **1|===] \\
\]

-----------------------------

That’s all folks!
5.2 ETM

In Fig. 2, a diagram of an ETM design is shown ($n_2n_2e.tm$). Note that, in our design, the automaton is always looping. All the transitions omitted halt the machine but, in theory, we cannot find this kind of transactions otherwise it will be an algorithmic error or a finite set!

Sigma=[01]
Delta=[]
---------------
Example using an ETM
| Name: concatenation of two binary numbers
| Sigma = {0, 1}
| $L = n_2n_2$ (n_2 is a binary number)
| Description: characteristic function of $L$
| Function: $f : Empty\_Set \to L$
| $f() = L$ enumeration
| Input: none
| Output: 00 11 1010 1111 100100 ... forever
| Date: October 18, 2011
| Version: 1.0
| UTM-version: 0.235711317192329313741
| Author: Copyright 2011 Jaume Rigau

Figure 2: Diagram for the $n_2n_2e.tm$ automaton.
5.3 DTM

We show in Fig. 3 a representation of a DTM design (\( n_2 n_2 d.tm \)). Note that, in order to simplify DTM-diagrams:

- \( \delta(q, \alpha) = (q, \beta, \omega) \) is represented by an edge from node \( p \) to \( q \) labeled with \( \alpha \rightarrow \beta | \omega \).

- All the transitions that are not represented go to a reject state neither writing any symbol nor moving any head (i.e., \( \alpha \rightarrow *|* \)).

- A set of transitions \( \{\alpha_1 \rightarrow \beta_1|\omega_1, \ldots, \alpha_n \rightarrow \beta_n|\omega_n\} \) is represented by \( \alpha_1 \cdots \alpha_n \rightarrow \beta_1 \cdots \beta_n|\omega_1 \cdots \omega_n \). Do not confuse this representation with multi-tape transitions.

\[ \Sigma = [01] \]
\[ \Delta = [2359] \]

Example using a DTM

- Name: concatenation of two binary numbers
- \( \Sigma = \{0, 1\} \)
- \( L = n_2 n_2 \) (\( n_2 \) is a binary number)
- Description: characteristic function of \( L \)
- Function: \( f : \Sigma^* \rightarrow \{\text{reject, accept}\} \)
- \( f(w) = \text{accept, if } w \in L \)
- \( = \text{reject, otherwise} \)
- Input: word \( w \) (e.g., 10111011)
- Output: accept / reject
- Date: October 18, 2011
- Version: 1.0
- UTM-version: 0.2357111317192329313741
- Author: Copyright 2011 Jaume Rigau

Fundamentals of Computing 20 Universal Turing Machine
Figure 3: Diagram for the $n_2n_2d.tm$ automaton.
Is it 00? (minimum word)
[start|0]=[0|*|>]
[0|0]=[00|*|>]
[00]=accept [*|*

It can only start with 1
Where is the half of the word?
[start|1]=[first_right|5|>]
[first_right|0]=[first_right|*|>]
[first_right|1]=[first_right|*|>]
[first_right|2]=[mark_right|*|<]
[first_right|3]=[mark_right|*|<]
[first_right| ]=[mark_right|*|<]
[mark_right|0]=[first_left|2|<]
[mark_right|1]=[first_left|3|<]
[first_left|0]=first_left|*|<
[first_left|1]=first_left|*|<
[first_left|2]=mark_left|*|>
[first_left|3]=mark_left|*|>
[first_left|5]=mark_left|*|>
[mark_left|0]=first_right|2|>
[mark_left|1]=first_right|3|>
[mark_left|2]=mark_left|*|>
[mark_left|3]=mark_left|*|>
[mark_left|5]=mark_left|*|>
[mark_right|0]=mark_left|*|>
[mark_right|1]=mark_left|*|>
[mark_right|2]=mark_right|*|>
[mark_right|3]=mark_right|*|>
[mark_right|5]=mark_right|*|>

In the first half: 0=2, 1=3, 1=5 (first cell)
In the second half: 0=0, 1=1
Matching (9 if it is ok)
[first_1|2]=[first_1|*|>]
[first_1|3]=[first_1|*|>]
[first_1|9]=[first_1|*|>]
[first_1|1]=left|9|<
[left|2]=left|*|<
[left|3]=left|*|<
[left|9]=left|*|<
[left| ]=which?|*|>]

Universal Turing Machine
5.4 NTM

In Fig. 4, a NTM-F design is represented \((n_2 \cdot n_2.n.tm)\).

---

That’s all folks!

5.4 NTM

In Fig. 4, a NTM-F design is represented \((n_2 \cdot n_2.n.tm)\).
Example using a NTM_F

Name: concatenation of two binary numbers
Sigma = \{0, 1\}
L = n_2 \cdot n_2 (n_2 is a binary number)
Description: characteristic function of L
Function: f : Sigma^* → Boolean
f(w) = 1, if w in L
= 0, otherwise
Input: word w (e.g., 1011011)
Output: 0 or 1
Date: October 21, 2011
Version: 1.0
UTM-version: 0.2357111317192329313741
Author: Copyright 2011 Jaume Rigau
Contact: jaume.rigau@udg.edu

Is it epsilon?
[start| **]= [halt| *0| ==]

Is it 00? (minimum word)
[start|0**]= [0| *0| >**]
[0|0**]= [00| **| >**]
[00| **]= [halt| *1| ==]

It can only start with 1
Where is the half of the word?
Probability:
In the state 'half' we conjecture if the current bit is the last bit of w.
We have 50% => p = 1/2 at each bit.
With |w| = n, we have a tree with n + 1 leaves:
a) i = 1..n => p = 1/(2^i)
b) i = n + 1 => 1/(2^n)
The sum of a) is 1 - 1/(2^n).
Plus the last leaf b), we have the unit.
If w is in L, the correct path is in the n/2 position => p = 1/(2^\lfloor n/2 \rfloor).
For example, if n = 6, we have 1 probability between 8 to accept the word.
[start|1**]= [half| *0| >=]
[half|0**]= [half|0*| >**]
[half|1**]= [half|1*| >**]
[half|0**]= [left|0*| >**]
[half|1**]= [left|1*| >**]

Going to first position
5.5 Another DTM

Here, in Fig. 5, we depicted a TM for \( L = \{ w \in \{a, b, c\}^* \mid w = a^ib^jc^i, i, j > 0 \} \).
You can see the corresponding code in \( a^i b^j c^i d.tm \) (TM directory).

5.6 Recursive Language

\( \Sigma = \{abc\} \)
\( \Delta = \{01\} \)

---
| Name: Chi_L |
| Sigma = \{a, b, c\} |
| L = a^nb^nc^n, n \geq 0 |
| Third of three examples about languages: |
| This is an example of Turing-decidable language |
| Out of NFA and PDA powers |
| Description: characteristic function of L |
| Function: \( f : \Sigma^* \rightarrow \text{Boolean} \) |
| \( f(w) = 1 \) = accept (flag -B), if \( w \) in \( L \) |
| \( = 0 \) = reject (flag -B), otherwise |
| Input: word \( w \) (e.g., abbaccabba) |
| Output: 0 or 1 |
| Date: October 3, 2011 |
| Version: 1.0 |
| UTM-version: 0.235711317192329313741 |
| Author: Copyright 2011 Jaume Rigau |
| Contact: jaume.rigau@udg.edu |
---

i) Epsilon case
\[ \text{init} *\text{=}\text{halt} 1\text{=} \]
\[ \text{init} @\text{=}\text{count_a} 0\text{=} \]
---

ii) Case symbol a
\[ \text{count_a}|\text{a}\text{=}\text{count_a}|\text{a}\text{=} \]
\[ \text{count_a}|\text{b}\text{=}\text{count_b}|\text{b}\text{=} \]
Figure 5: Diagram for the $a^{i}b^{j}c^{k}d^{l}$ automaton.
5.7 Context-Free Language

---

iii) Case symbol b
[count_b|ba*]=[count_b|b*|><*]
[count_b|c *]=[count_c|**|=>*]
---

iii) Case symbol c
[count_c|cb*]=[count_c|c*|>>*]
[count_c| *]=[halt|*1|===]
---

That’s all folks!

5.7 Context-Free Language

Sigma=[abc]
Stack=[$]
Final=[start|halt]

| Description: Chi_L |
| Sigma = {a, b, c} |
| L = a"nb"n, n >= 0 |
| Second of three examples about languages: |
| This is an example of PDA (i.e., Context-Free Language) |
| It is strictly included in Turing-decidable languages |
| Function: f : Sigma^* -> Boolean |
| f(w) = accept, if w in L |
| = reject, otherwise |
| Input: word w (e.g., abbacccabba) |
| Output: accept / reject |
| Date: October 19, 2013 |
| Version: 1.1 |
| UTM-version: 0.2357113171923293137414347535961 |
| Author: Copyright 2013 Jaume Rigau |
| Contact: jaume.rigau@udg.edu |

---

[start|..]=[a|$]
[a|a.]=[a|a]
[a|ba]=[b|.|]
[b|ba]=[b|.|]
[b|.$]=[halt|.|]
---

That’s all folks!

5.8 Regular Language

Sigma=[abc]
Final=[abba]

-----------------
| Description: Chi_L
| Sigma = {a, b, c}
| L = Sigma*abbaSigma*
| L = "exists the sequence 'abba'"
| Function: f : Sigma* -> Boolean
| f(w) = 1, if word in L
| = 0, otherwise
| Input: word w (e.g., abcaabbaccaabbac)
| Output: 0 or 1
| Date: November 7, 2012
| Version: 1.2
| UTM-version: 0.235711131719232931374134753961
| Author: Copyright 2011 Jaume Rigau
| Contact: jaume.rigau@udg.edu
| Design:
| Non-deterministic
| p(accept) = Sum(i in 1..K) 1 / 2^position(sequence_i)
| where K = number of abba sequences in w
| and position(sequence_i) = position of the first a of abba in w
| Example test: /test_infinite.tm.sh abba.nfa abcaabbaccaabbac
| K = 2 and p = 1/2^3 + 1/2^6 = 1/8 + 1/64 = 9/64 = 0.140625

------------------------------------------------------------------------

[prefix|a]=[prefix]
[prefix|b]=[prefix]
[prefix|c]=[prefix]
[prefix|a]=[a]
[a|b]=|ab|
[ab|b]=|abb|
[abb|a]=|abba|
[abba|a]=|abba|
[abba|b]=|abba|
[abba|c]=|abba|

-----------------
That’s all folks!

6 Version

6.1 Implementation

This documentation corresponds to the utm implementation

Version 0.235711131719232931374134753961
October 19, 2013
The version\textsuperscript{1} number is necessary for any reference and \textit{tm}-files. At present, a \texttt{utm} implementation\textsuperscript{2} is allocated in the server \texttt{bas.udg.edu}. Terminal direct access inside the campus is by login \texttt{mac}\textsuperscript{3} and going to folder \textit{UTM}\textsuperscript{4}. Remote access is by \texttt{ssh -l mac bas.udg.edu} or \texttt{sftp mac@bas.udg.edu}.

Happy computations!

6.2 License

\begin{quote}
\copyright \ 2009-2013 Jaume Rigau.

Permission is hereby granted to any person obtaining a copy of this software, and associated documentation files (the “Software”), for any non-profit, training, research, or educational purpose whatsoever – and not for any direct or indirect commercial purpose or advantage – without any fee or cost, to use, copy, create derivative works, publish, and distribute. The persons to whom the Software is furnished are subject to the following conditions:
- Any change in the received code shall be documented in this permission notice (date, author, and objective).
- The above copyright notice, permission, changes (if necessary), this sentence, and the below paragraph shall be included in all total or partial copies of the Software.
- Inform to the e-mail that appears in the cover.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
\end{quote}

\begin{flushright}
Machines take me by surprise with great frequency
\end{flushright}

\begin{flushright}
\textit{Alan Turing}
\end{flushright}

\textsuperscript{1}I apologize for any mistake —including my English— but comments, typos, and mainly, errors and \texttt{utm} bugs will be welcome. Do not hesitate to contact —see e-mail in the title. Thanks.

\textsuperscript{2}Remember to use the call \texttt{./utm} if the current directory is not inside the default search path.

\textsuperscript{3}The password is known in your course (or contact).

\textsuperscript{4}The default executable version \texttt{utm} is for Linux Redhat 64 bits (i.e., \texttt{bas} server OS) but you can also find a version for Readhat 32 bits and another for OS X 10.6 64 bits or later, called \texttt{utm.32} and \texttt{utm.64} respectively.