CS 5800 Notes: Turing Machines

based on T. A. Sudkamp [2]

1 Introduction: Computability

We will be mainly interested in algorithms or effective procedures for solving decision problems. Note that a decision problem can be related to a membership problem in a language. For example, the problem to determine whether a given natural number is a perfect square, has the following "language view":
for a given string $w \in \{a\}^*$, is $w \in L_{sq} = \{a^k \mid k \text{ is a perfect square}\}$?

An algorithm has the following characteristics:
– *mechanistic*: consists of a finite sequence of instructions, each of which can be executed without ingenuity or insight;
– *deterministic*: always produces the same computation when presented with an identical input;
– *complete*: delivers the correct result for each problem instance.

The Turing machine (TM), introduced by Alan Turing in 1936, can be used as a framework or model of computation in view of its effectiveness: the standard TM is mechanistic and deterministic; a TM that halts for every input is also complete. A non-deterministic version can be considered, which is however not more powerful than the deterministic TM, since a non-deterministic TM can be converted to a deterministic one.

Various formalisms have been used as a model of computation, based on expressing effective procedures by:
– rules that transform strings (Post systems, Markov systems, unrestricted grammars);
– evaluation of functions (partial recursive functions, lambda calculus);
– abstract computing machines (register machines, TM);
– programs in high-level languages.
The Church-Turing thesis by Alonzo Church in 1936 asserts that an effective computation in any algorithmic system can also be achieved by a suitably designed TM.

2 Standard Turing Machine

Definition 2.1. *A (standard)* **Turing machine** *is a quintuple* $(Q, \Sigma, \Gamma, \delta, q_0)$ *where* Q *is a finite set of states with* q_0 *as the start state,* Σ *a finite set of alphabet symbols,* $\Sigma \subseteq \Gamma - \{B\}$

where Γ *is a finite set of tape symbols and* B *is the blank symbol, and* δ *is the transition function, which is a partial function* $\delta : Q \times \Sigma \to Q \times \Sigma \times \{L, R\}$.

As a tape machine, the TM consists of a tape divided in tape squares or cells, a tape R/W head and a control mechanism. We adopt a model where the tape is bounded on the left and extends infinitely to the right. As an example, Fig. 1 shows the tape head scanning

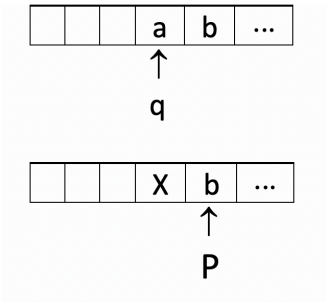

Figure 1: Sample move $\delta(q, a) = [p, X, R]$

a cell with tape symbol a in state q, and a transition where a is replaced by X and the tape head moves one square to the right, i.e., $\delta(q, a) = [p, X, R]$.

Envision the tape cells indexed starting at 0 from left to right. As another convention, the input string is written starting at square 1, and the tape head is positioned over square 0 at the beginning of a computation. Fig. 2 shows some (not all) intermediate configurations of a TM copy procedure that copies the string $u = abb$ to the right of the first blank after string u. Representing the original tape contents by BuB, the copy procedure yields $BuBuB$ (where the rightmost B is used to delineate the non-blank strings on the tape). The TM copies the symbols one by one, while it marks a and b in the original string by X and Y, respectively. When it is detected that no symbols are left to be copied, the X and Y symbols are replaced by the original characters as the tape head moves back to the left.

Note that the copy mechanism could be used, for example, to generate the product of two natural numbers written by strings x and y in unary notation on the tape. The string x could be copied y times (i.e., as many times as there are 1s in y). For $x = 3$ written as 111 on the tape, and $y = 2$ represented as 11, the final result on the tape would be $B111B11B111111B$. This helps showing the versatility of the Turing machine capabilities. It is held by the Church-Turing thesis that any effective (algorithmic) procedure can be implemented by a TM.

The TM transitions can be represented by a graph or by state transition table. Fig. 3 displays a state diagram of a COPY TM [2], where the transitions between states are labeled with their replacement of the symbol on the tape (from/to) and the move left (L) or right (R). The corresponding state transition table is given below, which has a row for each state and a column for each tape symbol (including the alphabet symbols and blank, B). The start state q_0 is indicated with an arrow ($\to$).

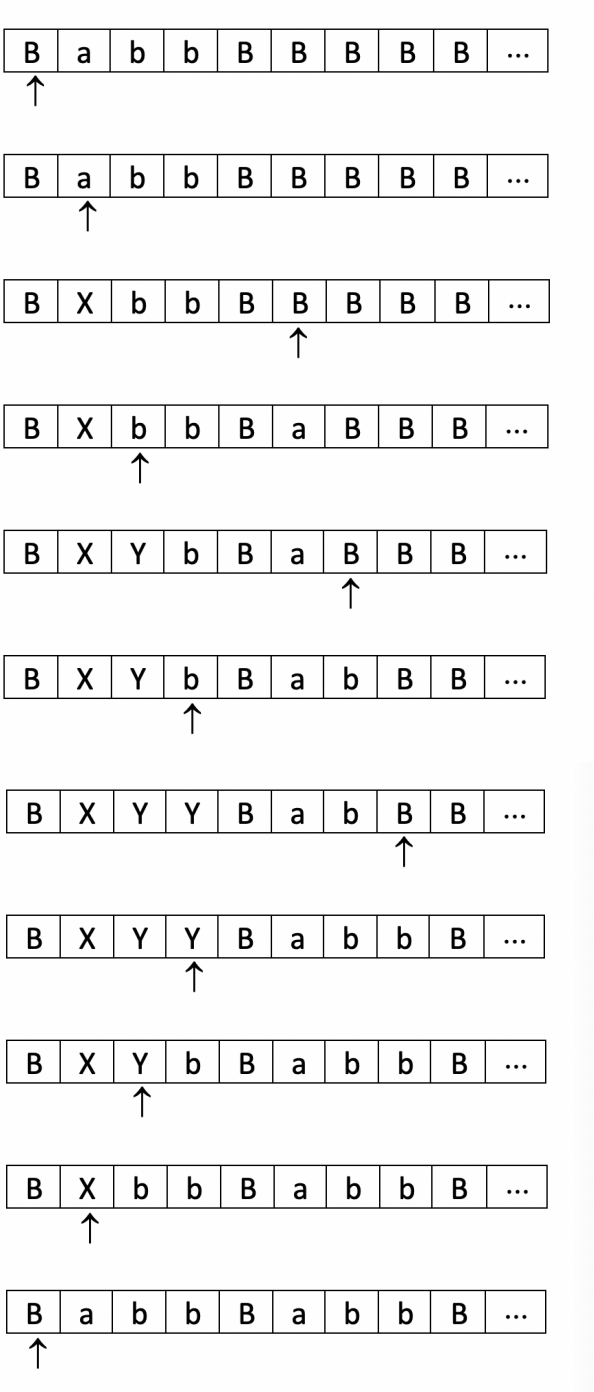

Figure 2: Intermediate configurations of copy TM computation

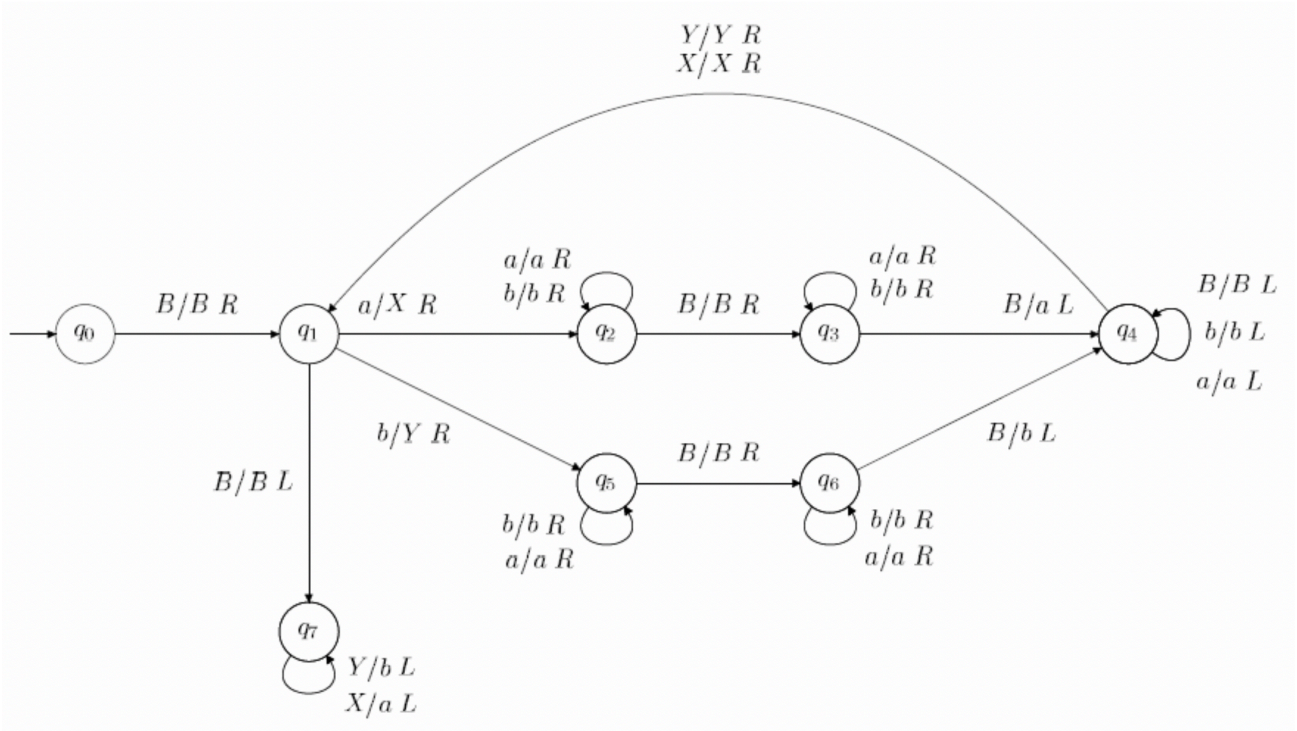

Figure 3: COPY TM [2]

δ	a	b	X	Y	B
$\rightarrow q_0$					q_1, B, R
q_1	q_2, X, R	q_5, Y, R			q_7, B, L
q_2	q_2, a, R	q_2, b, R			q_3, B, R
q_3	q_3, a, R	q_3, b, R			q_4, a, L
q_4	q_4, a, L	q_4, b, L	q_1, X, R	q_1, Y, R	q_4, B, L
q_5	q_5, a, R	q_5, b, R	q_6, B, R		
q_6	q_6, a, R	q_6, b, R			q_4, b, L
q_7			q_7, a, L	q_7, b, L	

A TM computation can be represented as a sequence of configurations, each of which contains the current non-blank region on the tape, with the state name written just before the position of the tape head. For the input string $u = abb$, the computation starts as:

$q_0BabbB \vdash Bq_1abbB \vdash BXq_2bbB \vdash BXbq_2bB \vdash BXbbq_2B \vdash BXbbBq_3B \vdash BXbbBq_4aB \vdash BXbbq_4BaB \vdash BXbq_4bBaB \vdash BXq_4bbBaB \vdash Bq_4XbbBaB \vdash BXq_1bbBaB \vdash BXYq_5bBaB \vdash \ldots$

This covers marking the symbol a of the input string u with X, copying the symbol a to the right of the blank (B) after string u, skip to the left over the string until X is found, and move one cell to the right where b is replaced by Y. Then the loop will start again, to copy the symbol b to the end of the copy of u generated so far, etc.

3 (Standard) TM language acceptor

Definition 3.1. *A (standard)* **TM language acceptor** *that* **accepts by final state** *is a six-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ where the first five elements of the tuple are as for the standard TM of Definition 2.1, and $F \subseteq Q$ is the set of accepting or final states. A string $u \in \Sigma^*$ is accepted by final state if M halts in a final state on input u. If M halts in a non-final state or halts abnormally [1] on input u, then u is rejected. For a string u that is not accepted it is also possible that M loops forever on input u. The set of all strings accepted by M is the* language $L(M)$ of M.

Definition 3.2. *The* **language** $L(M)$ *of the TM M is the set of all strings accepted by M. A language accepted by a TM is a* **recursively enumerable (r.e.)** *language.*

A TM accepting an r.e. language L is depicted on the left of Fig. 4. It halts (and accepts) for all $u \in L$, but for $u \notin L$ the TM may reject or may loop forever. If, for all $u \in L$, there is a TM that always halts and either accepts u (if $u \in L$), or rejects u (if $u \notin L$), then the r.e. language L is also *recursive*. This type of TM is shown on the right of Fig. 4, where 'Y' and 'N' mean 'Yes' (accepted) and 'No' (rejected), respectively.

Definition 3.3. *If there is a TM M for L that halts for every input in Σ^*, then L is a* **recursive** *language.*

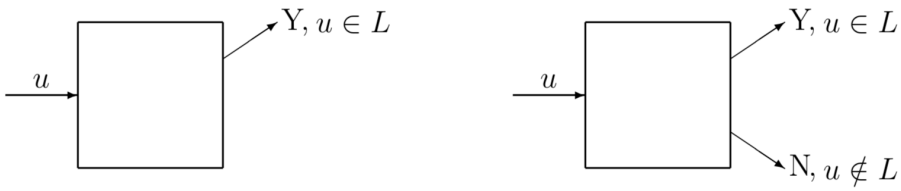

Figure 4: Left: TM for r.e. language L; Right: TM for recursive language L

The TM M with transition table listed below accepts the language

$$L(M) = (a \cup b)^* aa (a \cup b)^*$$

by final state. The start state (q_0) is indicated with an arrow ($\rightarrow$), and the final state (q_3) with a star ($*$).

[1] such as by falling off the left end of the tape

δ	a	b	B
$\rightarrow q_0$			q_1, B, R
q_1	q_2, a, R	q_1, b, R	
q_2	q_3, a, R	q_1, b, R	
$* q_3$			

The computation of M on the input string $u = abaabbab$ is given by

$q_0BabaabbabB \vdash Bq_1abaabbabB \vdash Baq_2baabbabB \vdash Babq_1aabbabB \vdash Babaq_2abbabB \vdash Babaaq_3bbabB$

The string u is accepted in state q_3 after only the part $abaa$ of u is processed.

On input string $u = aba$ the computation is

$q_0BabaB \vdash Bq_1abaB \vdash Baq_2baB \vdash Babq_1aB \vdash Babaq_2B$

The machine halts in non-final state q_2 as no transition is specified on the blank symbol from q_2, thus the input is rejected.

The language $L(M)$ is recursive as M will halt for every input. Of course, $L(M)$ (given by a regular expression) is regular. This corresponds to the fact that M only reads its input, from left to right, which corresponds with a finite state automaton.

Fig. 5 (from [2] displays a TM language acceptor M for $L(M) = \{a^ib^ic^i \mid i \geq 0\}$. This TM halts for every input $u \in \{a, b\}^*$, so $L(M)$ is recursive. In fact, the language can be written as $L(M) = L_1 \cup \{\lambda\}$ where $L_1 = \{a^ib^ic^i \mid i > 0\}$ is context-sensitive. The algorithm of M successively marks a symbol a with X, b with Y, and c with Z; then moves back to get the next a, b, c, etc. If that can be done throughout the input string it is accepted. The empty string (for $i = 0$) is also accepted.

4 Further TM variations

4.1 (Single) tape variations

Variations of the standard TM with respect to the tape include multi-track and two-way tapes. In spite of the apparent extension, however, language acceptors of these types have the same power as the standard TM, i.e., they accept precisely the r.e. languages.

4.1.1 Multi-track tape

The tape of a multi-track machine is divided into k tracks in general (track $1, 2, \ldots, k$). Labeling the tape positions as $0, 1, 2, \ldots$ from left to right, the input is written starting at position 1 on tape 1. There is only one R/W head that reads and writes an entire tape position at a time.

Fig. 6 shows a sample transition, $\delta(q, [x, y]) = [p, [s, t], R]$ for a 2-track machine. A general transition for a k-track TM with states $q_0, q_1, \ldots, q_n$ is

$$\delta(q_i, [x_1, x_2, \ldots, x_k]) = [q_j, [s_1, s_2, \ldots, s_k], d]$$

where $x_\ell, s_\ell \in \Gamma$ for $0 \leq \ell \leq k$; $q_i, q_j \in Q$ for $0 \leq i, j \leq n$; and the move $d \in \{L, R\}$.

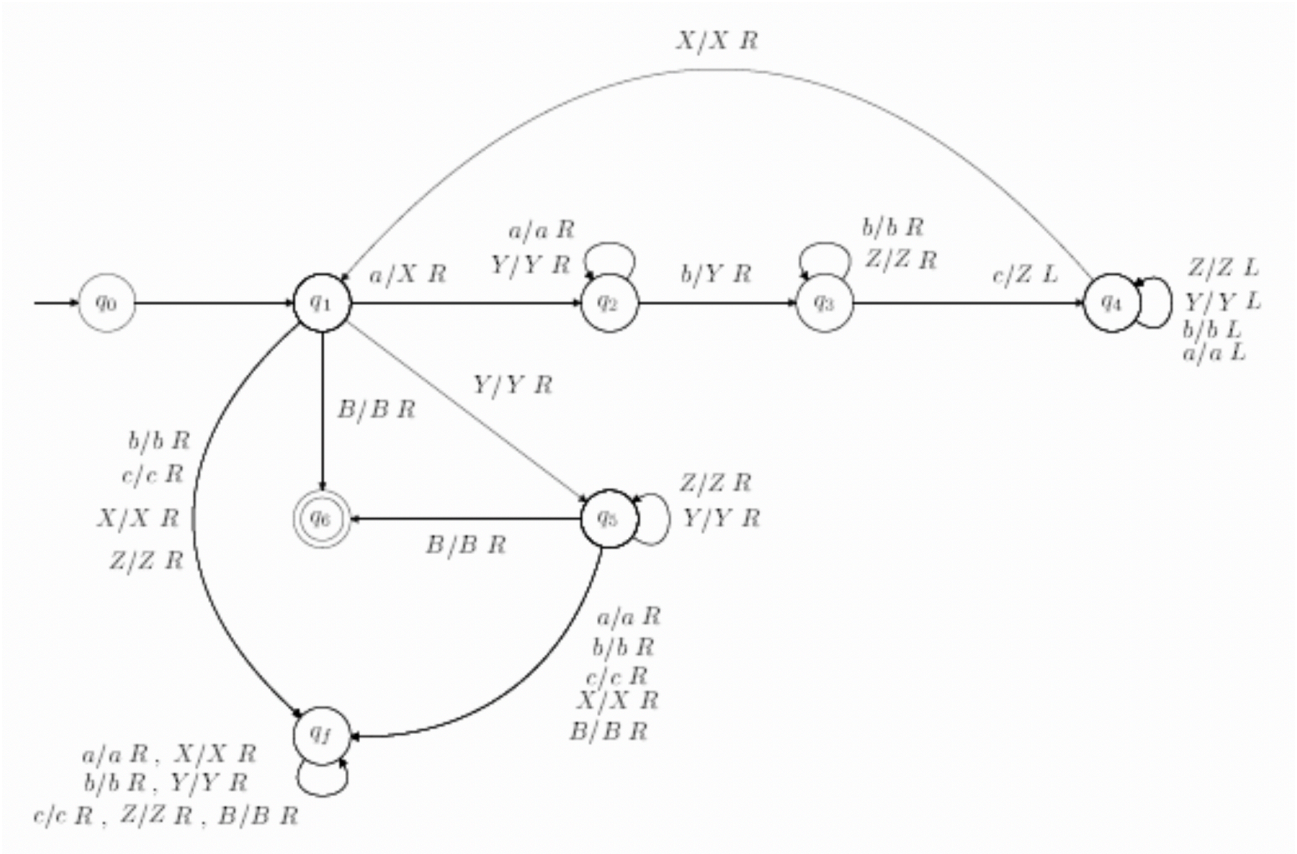

Figure 5: TM acceptor for the language $\{a^i b^i c^i \mid i \geq 0\}$ [2]

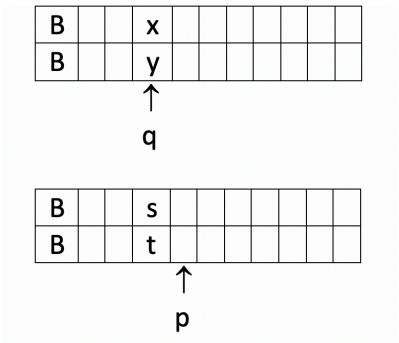

Figure 6: Transition of 2-track TM, $\delta(q, [x, y]) = [p, [s, t], R]$

Theorem 4.1. *A language L is accepted by a k-track TM $\Longleftrightarrow$ L is accepted by a standard TM (i.e., L is r.e.).*

4.1.2 Two-way tape

The tape of a two-way TM extends infinitely in both directions. Thus the input can be written anywhere on the tape. This is a common model, for example it is used in the text by Hopcroft, Motwani and Ullman [1].

Theorem 4.2. *A language L is accepted by a two-way TM $\Longleftrightarrow$ L is accepted by a standard TM (i.e., L is r.e.).*

Outline of proof: The $\Longleftarrow$ direction is trivial, since a standard TM computation can be executed on a one-way portion of the two-way tape bounded on the left (while the other portion of the two-way tape is neglected).
For the $\Longrightarrow$ direction, the two-way tape can be folded over into a two-track TM, which is equivalent to the standard TM according to Theorem 4.1.

4.2 Multi-tape TM

A multi-tape TM has a separate R/W head for each of its tapes. Fig. 7 shows a 2-tape TM transition $\delta(q, x, y) = [p; s, R; t, L]$.

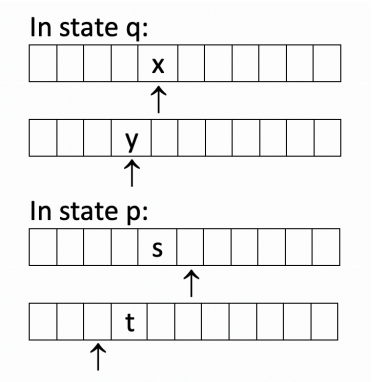

Figure 7: Transition of 2-tape TM, $\delta(q, x, y) = [p; s, R; t, L]$.

The general 2-tape transition is put in the form $\delta(q_i, x_1, x_2) = [q_j; s_1, d_1; s_2, d_2]$, for q_i to $q_j \in Q$, where x_i is replaced by s_i, followed by the move $d_i \in \{ L(=\text{ Left}), R(=\text{ Right}), S(=\text{ Stationary}) \}$ on tape $i(= 1, 2)$. 'Stationary', where the tape head does not move, is added for convenience; it could otherwise be implemented as a move Left followed by Right, or vice-versa.

Allowing multiple tapes does not affect the power of Turing machines:

Theorem 4.3. *A language L is accepted by a multi-tape TM $\Longleftrightarrow$ L is accepted by a standard TM (i.e., L is r.e.).*

Note also that mixed models are possible, such as a two-way multi-tape TM.

An advantage of multi-tape TMs is that they allow matching regions of symbols easily. For example, consider constructing a 2-tape TM for the language $\{a^i b^i c^i \mid i \geq 0\}$. The input string u is written on tape 1. The **algorithm** can proceed along the following lines:
If $u = \lambda$, it is accepted. Otherwise copy the a-region to tape 2 (if there is no a-region, reject the input). Then the tape head on tape 1 goes through the b-region of the input string (if there is no b-region, reject the input) while the tape head of tape 2 goes through its a-region, in order to match the b-symbols with the a-symbols. If the number of bs does not match the number of as, reject the input. Otherwise repeat the procedure to match the c-region with the a-region on tape 2.

For problems where different parts of a string need to be matched, these can be copied out to work tapes, so the tape heads can go through the regions simultaneously.

See also the example in the text [2] for the language $\{uu \mid u \in \{a, b\}^*\}$.

Another example is given for the language $L_{sq} = \{a^j \mid j \text{ is a perfect square}\}$. For a given input string $u = a^j$ this corresponds to solving a decision problem: to determine whether or not a number $j \in \mathbb{N}$ is a perfect square ($j = k^2$ for $k \in \mathbb{N}$).

The **algorithm** uses a 3-tape TM (see the text for a detailed description). Fig. 8 (from [2]) displays the input configuration, where the input string is written on tape 1. If $u = \lambda$, it

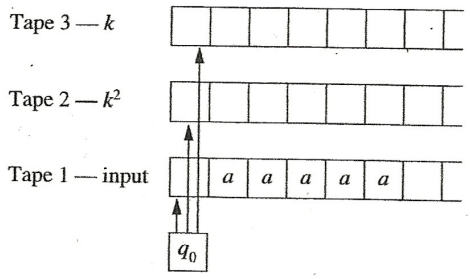

Figure 8: Input configuration of 3-tape TM for $L_{sq} = \{a^j \mid j \text{ is a perfect square}\}$ [2]

is accepted ($j = 0$). Otherwise, a sequence of steps is performed where the next square k^2 is generated on tape 2 and the corresponding k is kept on tape 3, for $k = 1, 2, \ldots$. By traversing the input string u on tape 1 and the string representing k^2 on tape 2 simultaneously, it is checked whether the length of the input string equals k^2. If it does, u is accepted. Otherwise, if k^2 generated on tape 2 becomes larger than the length of the input string, $k^2 > |u|$, then $|u|$ is strictly between two perfect squares, so it is rejected.

The steps for $k = 1, 2, 3$ are shown in Fig. 9 [2] for the input string $u = a^5$.
$\underline{k = 1}$: In this step, X is written on tape 2. the length of the input string is compared to that of X on tape 2. It is found that $|u| > |X| = 1$. Note that tape 3 is still blank (cf., the input configuration of Fig. 8). Then tape 3 is updated to contain X.
$\underline{k = 2}$: The string of length k^2 with $k = 1$ is updated on tape 2 to have length $(k + 1)^2 = k^2 + 2k + 1 = 4$. This is done by appending two copies of X from tape 3 to the string on tape 2, and incrementing by one X. The length of the input string is compared to that of

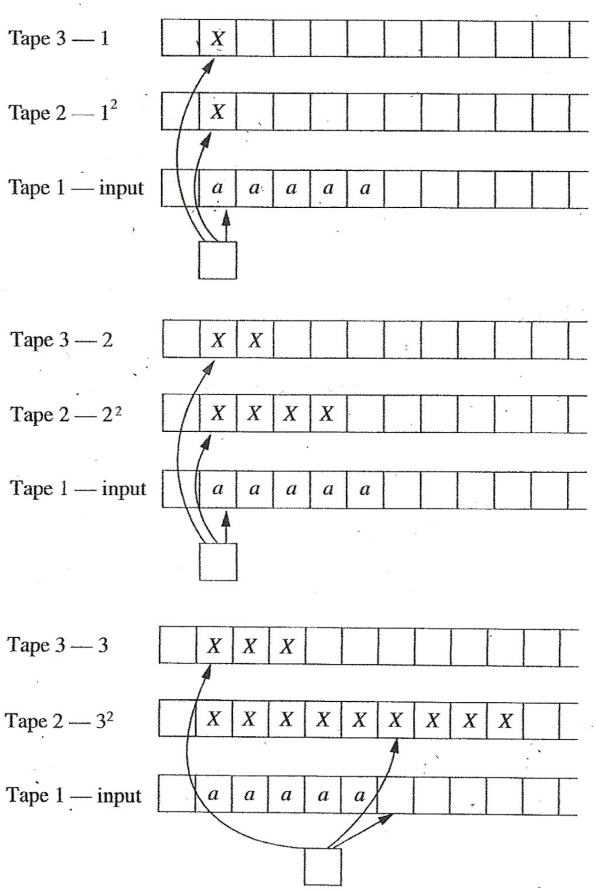

Figure 9: Steps for $k = 1, 2, 3$ of 3-tape TM to check if $|u| = k^2$ [2]

$XXXX$ on tape 2. It is found that $|u| > |XXXX| = 4$. Then the tape 3 string is incremented by one X, so it now contains XX.
$\underline{k = 3}$: The string of length k^2 with $k = 2$ is updated on tape 2 to have length $(k+1)^2 = k^2 + 2k + 1 = 9$. This is done by appending two copies of XX from tape 3 to the current string $XXXX$ on tape 2, and incrementing by one X, yielding X^9. The length of the input string is compared to that of X^9 on tape 2. It is found that $|u| < |X^9| = 9$. Therefore $4 < |u| < 9$ and the input string u is rejected.

4.3 Non-deterministic TM

Adding non-determinism does not increase the power of TMs.

Theorem 4.4. *A language L is accepted by a non-deterministic TM $\iff$ L is accepted by a standard TM (i.e., L is r.e.).*

Fig. 10 [2] gives an example of a one-tape non-deterministic TM, for the language L over $\Sigma = \{a, b, c\}$ of strings containing c preceded or followed by ab. In particular, three transitions are possible on reading c at state q_1.

As for non-deterministic finite state machines, a string will be accepted by a non-deterministic TM if there is a computation (i.e., the string traces a path) from the initial to

a final state of the TM.

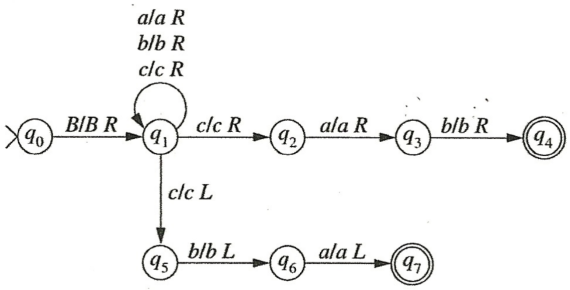

Figure 10: Non-deterministic TM [2]

4.4 TM enumerator

This type of TM lists (enumerates) all strings of a language on its tape. The procedure continues forever for an infinite language. As an example, a TM enumerator for the language $L = \{a^i b^i c^i \mid i \geq 0\}$ could list the strings of L on its tape as:

$$B\#\#abc\#aabbcc\#aaabbbccc\#\ldots$$

Here # is used as a separator between the strings. Note that ## denotes the empty string in between the separators. At any time, the machine is either in the process of listing a string, or has just finished listing a string.

Theorem 4.5. *A language L can be enumerated by a TM $\iff$ L is r.e.*

References

[1] Hopcroft, J. E., Motwani, R., and Ullman, J. D. *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, 2nd edition, 2001. ISBN 0-201-44124-1.

[2] Sudkamp, T. A. *An Introduction to the Theory of Computer Science − Languages and Machines.* Pearson, Addison Wesley, 3rd edition, 2006. ISBN 0-321-32221-5.