Teoria da Computação
2018/2

Álvaro Moreira
afmoreira@inf.ufrgs.br

Instituto de Informática
Universidade Federal do Rio Grande do Sul
Porto Alegre, Brasil
http://www.inf.ufrgs.br

Contents

Contents

Turing Machines - Intuition

1. Proposed by Alan Turing in 1936 a TM is a model of general purpose computer
2. It uses an **infinite tape** as its unlimited memory
3. Initially the tape has only the input string and its is blank in all other places
4. Turing analysed how a person equipped with a writing instrument, makes calculations with a sheet of paper organised in squares. This activity consist of a sequence of simple operations as:
   - read symbol of a square
   - write a symbol in a square
   - move its attention to a different square
   - when a satisfactory representation of the desired answer is obtained, the person concludes its calculations

Turing Machine: Formal Definition

Definition: A **Turing Machine** is a 7-tupla: $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where Q, Σ, and Γ are all finite sets and

- Q is a set of states
- Σ is the input alphabet not containing the blank symbol $\sqcup$
- Γ is the tape alphabet, where $\sqcup \in \Gamma$, and $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function
- $q_0 \in Q$ is the start state
- $q_{accept} \in Q$ is the accept state, and
- $q_{reject} \in Q$ is the reject state, $q_{accept} \neq q_{reject}$

Turing Machine: transition function

- The heart of a Turing Machine is its transition function δ.
- It determines how the TM gets from one step to the next.
- The transition function has the following functionality:

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$$

- When a TM is in state q and the head is over tape square with a symbol a and if δ is such that $\delta(q, a) = (r, b, L)$, for example, that means the TM goes to state r, replaces the symbol a with b, and moves the head one square to the left

Turing Machine: computation

A TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ computes as follows:

- M starts with input $w = w_1 w_1 \dots w_n \in \Sigma^*$ on the leftmost squares of the tape
- all the tape squares to the right of w are filled with blank $\sqcup$
- the head starts over the leftmost square of the tape
- once started, M proceeds as defined by δ
- if δ instructs to move M's head to the left of the tape's left-hand end, the head stays in place
- the computation continues until M reaches either q_{accept} or q_{reject}
- if neither occurs M goes on forever

Turing Machine configurations

As a TM M computes changes occur in the (i) current state, (i) current tape contents, and (iii) current head position

Given a state q and strings u and v over the tape alphabet a TM **configuration** is written as uqv, and it represents that:

- the string uv is in the tape
- the TM is in state q, and
- the head is positioned over the first symbol of string v

A configuration like $1011q_701111$ represents that

- the string 101101111 is at the left end of the tape, and all the other squares are blank
- the current state is q_7, and
- the head is positioned over the second, from left to right, symbol 0

Turing Machine: change of configuration

For a, $b \in \Gamma$, and u, $v \in \Gamma^*$, and $q_1, q_j \in Q$, if the current configuration of a TM M is

$$uaq_ibv$$

and the function δ is such that

$$\delta(q_i, b) = (q_j, c, L)$$

then the next configuration is

$$uq_jacv$$

If the current configuration is q_ibv and $\delta(q_i, b) = (q_j, c, L)$ then that configuration changes to q_jcv (the head does not move in this case)

Configurations

The **start configuration** of M on input w is the configuration q_0w, indicating that M is at the start state q_0 with its head over the leftmost square of the tape

An **accept configuration** of M is any configuration with the state q_{accept}

A **reject configuration** of M is any configuration with the state q_{reject}

Both accept and reject configurations are **halting configurations** and do not lead to other configurations.

A TM M **accepts** input w if a sequence of configurations $C_1, C_2, \ldots, C_k$ exists, where;

1. C_1 is the start configuration of M on input w
2. each C_i yelds C_{i+1}, and
3. C_k is an accepting configuration

Turing Machines and languages

The collection of strings that a TM M accepts is **the language of M** or **the language recognized by M**, written L(M).

Definition: *Call a language* ***Turing-recognizable*** *if some Turing Machine recognizes it.*

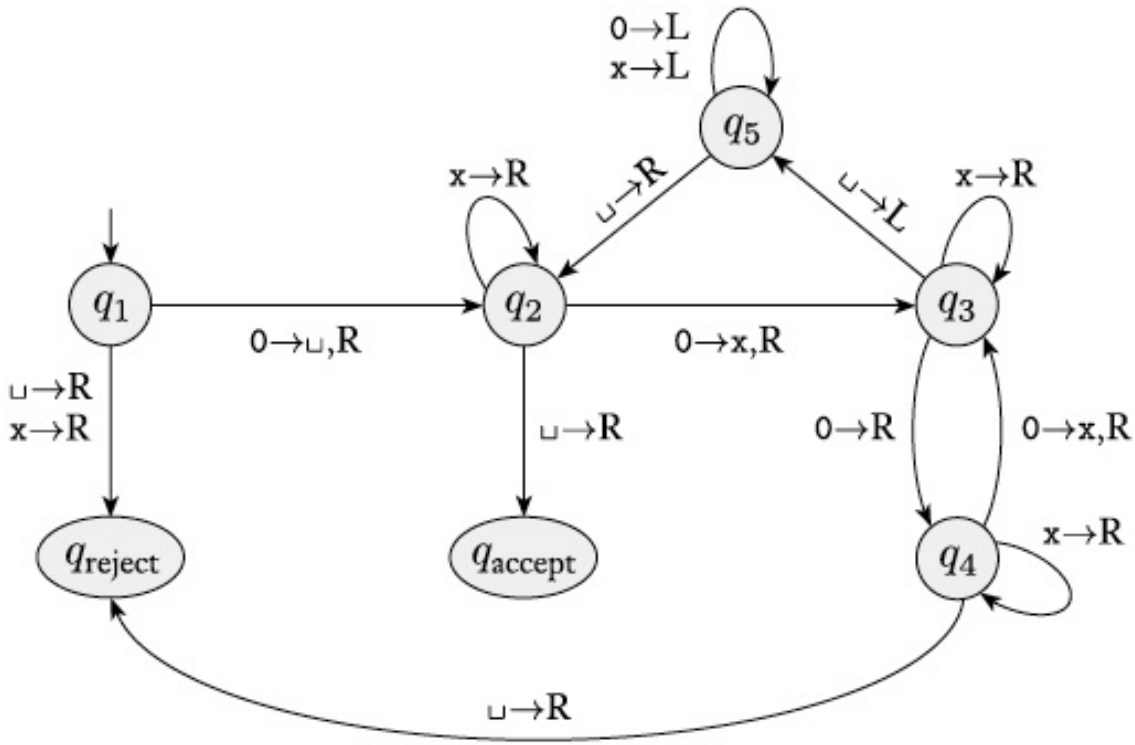
Observe that a TM that loops, given a string w, is failing to accept it (i.e. it is failing entering state q_{accept}). But we cannot distinguish if it entered in a loop or if it is just taking a long time.

For this reason we sometimes prefer machines that always halt. These machines are called **deciders**.

Definition: *Call a language* ***Turing-decidable*** *or simply* ***decidable*** *if some Turing machine decides it.*

Example of Turing Machine

Example: A Turing machine M_2 that decides $A = \{0^{2^n} \mid n \geq 0\}$, the language of all strings of 0s whose length is a power of 2.



Exercise: Write down the formal definition of the TM of the diagram above and trace its execution on some strings that belong to the language and others that don't.

Variants of Turing Machines

There are several alternative definitions of Turing machines

The original model and all its variants have the same power, i.e., they all recognize the same class of languages

We call this invariance to certain changes in the definition **robustness**

Consider, variant: besides moving the tape head to the left (L) or to the right (R) the transition function can also instruct the head to stay on the same position (S)

Observe that this modification does not add to the power of the original definition since we can simulate *stay put* (S) with a transition that moves right (R) immediately followed by a transition to move left (L)

We consider two variants: TMs with multiple tapes and nondetermistic TMs

Turing Machines with Multiple Tapes

Definition: A **Turing machime with** k **fitas** has k unidirectional tapes with k read/write tape heads.

$$\begin{array}{lll} \text{tape}_1: & \overset{\nabla}{a}\ a\ b\ a\ b & \cdots \\ \text{tape}_2: & b\ b\ \overset{\nabla}{b}\ a\ a\ a & \cdots \\ \vdots & & \\ \text{tape}_k: & a\ a\ b\ \overset{\nabla}{b} & \cdots \end{array}$$

Initially the input is on tape 1 and the others start out blank.
The transition function δ has the following signature

$$\Pi : (Q \times \Gamma^k) \rightharpoonup (Q \times \Gamma^k \times \{L, R, S\}^k)$$

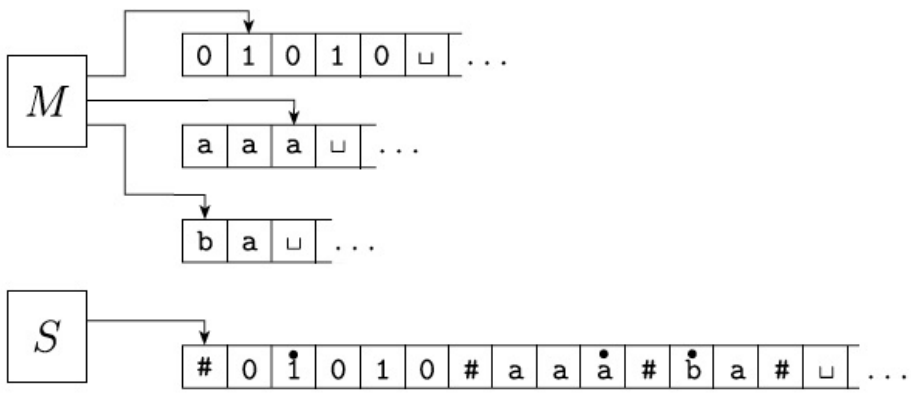
$\delta(q_1, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, \overbrace{L, R, \ldots L}^{k})$ means that if the machine is in state q_i and heads 1 to k are reading symbols a_1 to a_k then the machine goes to state q_j , writes symbols $b_1, \ldots b_k$ and directs each head to move left, right or to stay put

Multitape Turing Machines

Two machines are equivalent if they recognize the same language.

Theorem: Every multitape Turing machine has an equivalent single tape.

Proof: Let M be a TM with k tapes over the alphabet Γ. We will build a single tape TM S over the alphabet $\Gamma \cup \{\sharp\} \cup \dot{\Gamma}$, where the new symbol $\sharp$ separates the contents of the different tapes,and the symbols in $\dot{\Gamma}$ are those of Γ with a dot above to mark the place where the head of that tape would be.



Read the proof of **Theorem:** 3.13 in Sipser's book to see how S can be constructed from M.

Nondeterministic Turing Machines

A nondeterministic Turing machine can proceed according to several possibilities.

The transition function δ of a ND TM has the following signature:

$$\delta : Q \times \Gamma \to \mathscr{P}(Q \times \Gamma \times \{L, R\})$$

The computation of a ND TM is a tree whose branches correspond to the different possibilities for the machine.

If some branch of the computation leads to an accept state then the ND TM accepts its input.

Theorem: Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

Proof: Read the proof of **Theorem:** 3.6 of Sipser's book

Algorithms

Algorithms are commonplace in everyday life. They also play important role in mathematics

Ancient mathematical literature describes algorithms for several tasks such as finding prime numbers and great common divisors.

Despite that, the notion of algorithm was formally defined only in the 20th century

Such a **formal definition of algorithm** is essential to prove the **non existence** of an algorithm to solve a problem

Hilbert's 10th Problem

In 1900, the German mathematician David Hilbert delivered a famous talk at the International Congress of Mathematics in Paris.

He identified 23 problems and posed them as a challenge for the century. The 10th problem is concerned to algorithms.

A **polynomial** is a a sum of terms where each term is the product of certain variables and a constant called **coefficient**. Example:

$$6x^3yz^2 + 3xy^2 - x^3 - 10$$

is a polynomial with four terms over variables x, y, and z. Here we consider only integer coefficients.

A **root** for the polynomial is an assignment of values to its variables so that the value of the polynomial is 0.

The root of the example above is $x = 5$, $y = 3$, and $z = 0$. This is an **integral root**. Some polynomial have integral root and others do not.

Church-Turing Thesis

Hilbert's 10th problem was to devise an algorithm to decide whether a polynomial has an integral root

We know now that this is an algorithmic **unsolvable** problem. But for the mathematician of that time to reach that conclusion only with an intuitive notion of algorithm would have been impossible

The intuitive notion of algorithm was enough to propose algorithms but not for **proving the non existence** of algorithmic solutions.

This formal definition came in 1936 with Alonzo Church's Lambda Calculus and with Alan turing's *a-machines* which later became know as Turing Machines.

These two definitions were proved to be equivalent, and the equivalence between the informal notion and the precise definition became known as the **Church-Turing Thesis**

Hilbert 10th Problem

In 1970 is was proved that no algorithm exists for deciding whether a polynomial has integral roots.

Here is Hilbert's 10th Problem in our terminology: consider the following language

$$D = \{\langle p \rangle \mid p \text{ is a polynomial with an integral root}\}$$

Hilbert's 10th Problem asks in essence if the language D above is **decidable** (it is not!)

But D is Turing-recognizable! (recall that to be Turing-recognizable does not require terminatiom of the TM that makes the recognition).

Hilbert 10th Problem

To see how lets consider polynomial with a single variable such as $2x^2 + x - 7$.

$$D_1 = \{\langle p \rangle \mid p \text{ is a polynomial over } x \text{ with an integral root}\}$$

The following TM M_1 recognizes D_1:

M_1 = "On input $\langle p \rangle$ where p is a polynomial over the variable x evaluates p successively with values $0, 1, -1, 2, -2, 3, -3 \ldots$. If at any point in this process, if p evaluates to 0 stop with *accept*"

For a polynomial with more than one variable we can present a similar TM that recognizes D by just trying all combination of integer values for its variables.

It is also possible to define a TM that **decides** the language D but in 1970 it was proved that no such a decider exists for a polynomial with multiple variables.

Machines as Strings

- So far each TM is "built"to compute only a specific function
- A TM can be represented as a **string**. Just write the description of the TM on paper, and then encode this description as a sequence of zeros and ones
- This string can be given as input to another TM
- Since the behaviour of a TM is determined by its transition function δ we will use the transition function as the basis for our encoding.

Exercise: Define an encoding as a string in $\{0,1\}^*$ for the TM M_{PAL}

The Universal TM (I)

- That (the encoding of) a TM can be the input for another TM motivated the invention of the *general-purpose* computer
- A *general-purpose* machine is a **single** machine which, loaded with the appropriate program, can compute any arbitrary task
- Turing was the first to observe that general-purpose computers are possible by showing a Universal Turing Machine
- A Universal Turing Machine can simulate the execution of any other TM M, given the encoding of M as input

The Universal TM (II)

- We take this notion for granted, but Turing's idea was at that time disruptive and counter-intuitive
- A universal TM, has a fixed alphabet, a fixed number of states and of tapes
- But it can simulate any other Turing Machine, including those with a larger alphabet, a larger number of states, and a larger number of tapes !!!
- Crucial to this counter-intuitive capacity is the notion of encoding

The Universal TM (III)

It is convenient that an encoding scheme satisfies the following two properties:

- Every string in $\{0,1\}^*$represents some TM (strings which are not valid encodings can be mapped to some trivial TM that halts immediately)
- Every TM can be encoded by infinitely many strings (also easy to ensure by just adding arbitrary number of 1s at the end of the encoding, which are ignored)

Note: there is nothing "deep"about these two properties. They just make some results simpler to state, such as the following theorem)

The Universal TM (IV)

In the following, M_α denotes the TM represented by string $\alpha \in \{0,1\}^*$.

Theorem: 1.9 *There exists a TM* $\mathcal{U}$ *such that for every* x, $\alpha \in \{0,1\}^*$, $\mathcal{U}(\alpha, x) = M_\alpha(x)$,
Moreover, if M_α *halts on input* x *within* T *steps then* $\mathcal{U}$ *halts on input* α *and* x *within* $C\ T\ \log T$ *steps where* C *is a number independent of* $|x|$ *and depending only on* M_α *alphabet's size, number of tapes, and number of states*

This theorem is saying that the universal TM can simulate/interpret any (dedicated) TM. It also says that there is a slowdown in the running time of the universal TM when compared to the running time of a dedicated TM

The Universal TM (V)

The input tape, the 1st work tape, and the output tape of the TM $\mathcal{U}$ will have the same content of corresponding tapes of a TM M being simulated/interpreted by $\mathcal{U}$. The 2nd work tape of $\mathcal{U}$ will have the string that represents the enconding of the TM M. The 3rd work tape will register the current state of the TM M.