Sapienza
Università di Roma
Dipartimento di Informatica

FOUNDATIONS OF COMPUTER SCIENCE
LECTURE 9: Turing machines

Prof. Daniele Gorla

Acceptors for non-context-free languages

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Up-to now:

Chomsky hierarchy

Regular languages	*Regular Grammars (Type 3)*	*DFA/NFA*
CF languages	*CF grammars (Type 2)*	*(non-det)PDA*
CS languages	*CS grammars (Type 1)*	*????*
RE languages	*Type 0 grammars*	*????*

In this class we want to complete this table in the setting of non-C.F. languages and provide acceptors (i.e., machines similar to automata) for the last 2 kinds of languages.

Idea:

- DFA/NFA: finite memory
- PDA: infinite memory, accessible only in a LIFO way
- Linear Bounded Automata: finite memory, accessible in any way → CS languages
- Turing machines: infinite memory, accessible in any way → RE languages

Since LBA are a special case of TM, we start from this computational model

Turing machines, informally (1)

- First proposed by Alan Turing in 1936
- Similar to finite automata but with an unlimited and unrestricted memory
- It is a much more accurate model of a general purpose computer (a TM can do everything that a real computer can do, and vice versa)
- It uses an infinite tape as memory, with a tape head that can read and write symbols and move around on the tape
- Initially the tape contains only the input string and is blank everywhere else
- It may read and write information on the tape by moving its head back and forth over it
- It continues computing until it decides to produce an output (*accept* or *reject* states)
- If it doesn't enter an accepting or a rejecting state, it will go on forever, never halting

Main features of TM:

1. It can both write on the tape and read from it (like PDA)
2. The head can move both to the left and to the right (different from PDA)
3. The tape is infinite (like PDA)
4. The rejecting and accepting states take effect immediately (different from DFA/NFA/PDA)

Turing machines, informally (2)

As an intuitive example, let's describe how a TM accepts $\{w\#w \mid w \in \{0,1\}^*\}$

separator

→ adding # is just for simplifying the description

→ also with # this language can be proved to be non-C.F.

On input string w:

- Zig-zag across the tape to corresponding positions on either side of the # symbol to check whether these positions contain the same symbol
- If they do not, or if no # is found, reject
- Cross off symbols as they are checked to keep track of which symbols correspond
- When all symbols to the left of the # have been crossed off, check for any remaining symbols to the right of the #
- If any symbols remain, reject; otherwise, accept

Turing machines, informally (3)

EXAMPLE: accepting 011000#011000

read the 1st char after #, if it's the same go on, else reject

0 1 1 0 0 0 # 0 1 1 0 0 0 ⊔ . . .

read '0'

x 1 1 0 0 0 # 0 1 1 0 0 0 ⊔ . . .

mark it as read and remember it

x 1 1 0 0 0 # x 1 1 0 0 0 ⊔ . . .

x 1 1 0 0 0 # x 1 1 0 0 0 ⊔ . . .

x x 1 0 0 0 # x 1 1 0 0 0 ⊔ . . .

x x 1 0 0 0 # x x 1 0 0 0 ⊔ . . .

do the same until the string runs out and it's all cancelled

... x x x x x x # x x x x x x ⊔ . . . → ACCEPT

the space is the empty character

EXAMPLE: rejecting 011000#111000

0 1 1 0 0 0 # 1 1 1 0 0 0 ⊔ . . .

x 1 1 0 0 0 # 1 1 1 0 0 0 ⊔ . . . → REJECT

as '0' was read before # and '1' was read after #

EXAMPLE: rejecting 011000#0

0 1 1 0 0 0 # 0 ⊔ . . .

x 1 1 0 0 0 # x ⊔ . . .

x 1 1 0 0 0 # x ⊔ . . .

x x 1 0 0 0 # x ⊔ . . . → REJECT

1 is not space, so the strings are different

EXAMPLE: rejecting 0#011000

2nd string is done before the 1st

0 # 0 1 1 0 0 0 ⊔ . . .

x # x 1 1 0 0 0 ⊔ . . .

x # x 1 1 0 0 0 ⊔ . . . → REJECT

we read #, so the 1st string is done before the second

Turing machines, formally

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

DEFINITION

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

1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** $\sqcup$,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $q_0 \in Q$ is the start state,
5. $q_{accept} \in Q$ is the accept state,
6. $q_{reject} \in Q$ is the reject state, where $q_{reject} \neq q_{accept}$,
7. $\delta : (Q \setminus \{q_{accept}, q_{reject}\}) \times \Gamma \to Q \times \Gamma \times \{L,R\}$ is the transition function.

- M receives its input $w \in \Sigma^*$ on the leftmost squares of the tape, and the rest of the tape is blank (since $\sqcup \notin \Sigma$, the first $\sqcup$ on the tape marks the end of the input)
- The head starts on the leftmost square of the tape
- Then, the computation proceeds according to the rules described by the transition function
- If M ever tries to move its head to the left off the left-hand end of the tape, the head stays in the same place for that move, even though the transition function indicates L
- The computation continues until M enters either the accept or reject states, at which point it halts
- If neither occurs, M goes on forever

Configurations for TMs

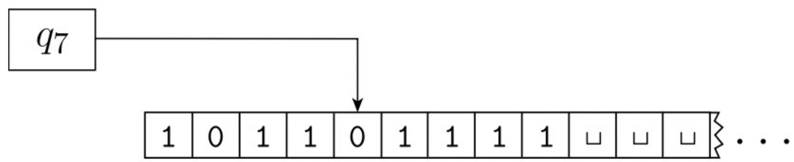
Computations rely on the current state, tape contents, and head location

A setting of these three items is called a ***configuration***

For $q \in Q$ and $u, v \in \Gamma^*$, we write uqv for the configuration where   * is the operation, every combination of Gamma

- the current state is q
- the current tape contents is uv we don't know past states
- the current head location is the first symbol of v
- the tape contains only blanks following the last symbol of v.

EXAMPLE: $1011q_701111$ represents the configuration



The ***start configuration*** of M on input w is q_0w

An ***accepting configuration*** is any configuration with state q_{accept}

A ***rejecting configuration*** is any configuration with state q_{reject}

Accepting and rejecting configurations are referred to as ***halting configurations***.

Computations and languages of TMs

Let $a,b,c \in \Gamma$, $u,v \in \Gamma^*$, and q_i, $q_j \in Q$. We say that

- configuration $ua\, q_i\, bv$ ***yields*** configuration $u\, q_j\, acv$ if $\delta(q_i, b) = (q_j, c, \mathrm{L})$
- configuration $ua\, q_i\, bv$ ***yields*** configuration $uac\, q_j\, v$ if $\delta(q_i, b) = (q_j, c, \mathrm{R})$
- configuration $q_i\, bv$ ***yields*** configuration $q_j\, cv$ if $\delta(q_i, b) = (q_j, c, \mathrm{L})$ already at the leftmost it can be, so b gets replaced w/o moving really
- configuration $q_i\, bv$ ***yields*** configuration $c\, q_j\, v$ if $\delta(q_i, b) = (q_j, c, \mathrm{R})$

in every case b is totally REPLACED

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

1. C_1 is the starting configuration of M on input w,

2. each C_i yields C_{i+1}, and

3. C_k is an accepting configuration. basically ends in an accepting configuration

The collection of strings that M accepts is ***the language of*** M, denoted $L(M)$.

The languages accepted by a TM are called ***recursively enumerable*** or ***semi-decidable.***

→ a TM can fail to accept an input by entering q_{reject} or by looping

→ a ***decider*** is a TM that always halts (by either accepting or rejecting in finite time)

The languages accepted by deciders are called ***recursive*** or ***decidable***.

Example: a TM for language $\{w\#w \mid w \in \{0,1\}^*\}$

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

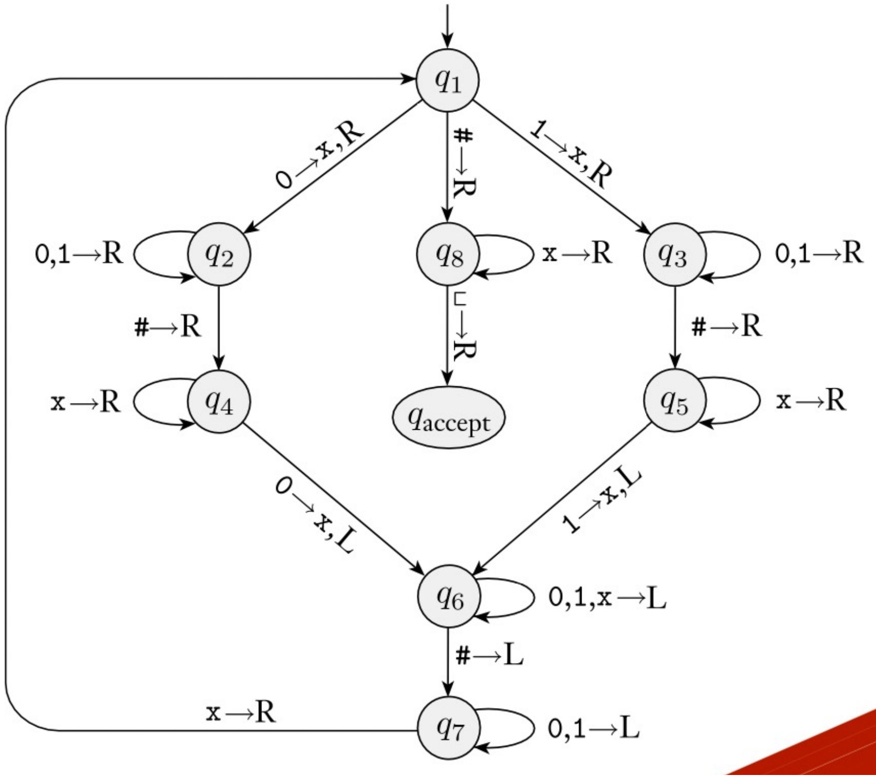
The TM behaves like we informally described before, i.e. $M = (Q, \Sigma, \Gamma, \delta, q_1, q_{accept}, q_{reject})$, with

- $Q = \{q_1, \ldots, q_8, q_{accept}, q_{reject}\}$,
- $\Sigma = \{0,1,\#\}$
- $\Gamma = \{0,1,\#,x,\sqcup\}$.
- δ is described with a state diagram, where we label an arrow from q to q'
  - with "$a \longrightarrow b$, X" , whenever $\delta(q, a) = (q', b, X)$, for $a,b \in \Gamma$ and $X \in \{L,R\}$ a changes into b
  - with "$a \longrightarrow X$" , whenever $\delta(q, a) = (q', a, X)$, for $a \in \Gamma$ and $X \in \{L,R\}$ a stays the same
  - with "$a,b \longrightarrow X$" , whenever there are two parallel arrows from q to q', one labeled with "$a \longrightarrow X$" and the other labeled with "$b \longrightarrow X$"

a and b lead to the same state

action according to point 2 is taken

To simplify the figure, we don't show q_{reject} and the transitions going there (these occur implicitly whenever a state lacks an outgoing transition for a particular symbol).



Variants of TMs: Multitape

- A ***multitape Turing machine*** is an ordinary Turing machine with several tapes (say, $k > 1$)
- Each tape has its own head for reading and writing.
- Initially the input appears on tape 1, and the others start out blank.
- The transition function allows for reading, writing, and moving the heads on some or all tapes simultaneously. Formally, it is

$$\delta : (Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\}) \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{\mathrm{L,R,S}\}^k$$

The expression

$$\delta(q, a_1, \ldots, a_k) = (q', b_1, \ldots, b_k, \mathrm{X}_1, \ldots, \mathrm{X}_k) \text{ , for } \mathrm{X}_i \in \{\mathrm{L,R,S}\}$$

means that, if the machine is in state q and head i (for all $i = 1,\ldots, k$) is reading symbol a_i, the machine goes to state q', writes symbol b_i on tape i (for all $i = 1,\ldots, k$), and directs head i to move left, right or stay, as specified by X_i.

REMARK: 'S' is needed whenever we don't want to use a tape, e.g. tape i: in this case, we shall simply have $b_i = a_i$ and $\mathrm{X}_i = \mathrm{S}$

Multitape TMs vs single tape TMs

Thm.: Every multitape TM M has an equivalent single-tape TM.

Proof

Say that M has k tapes.

We build a single-tape TM S that simulates the k tapes by storing them on its single tape.

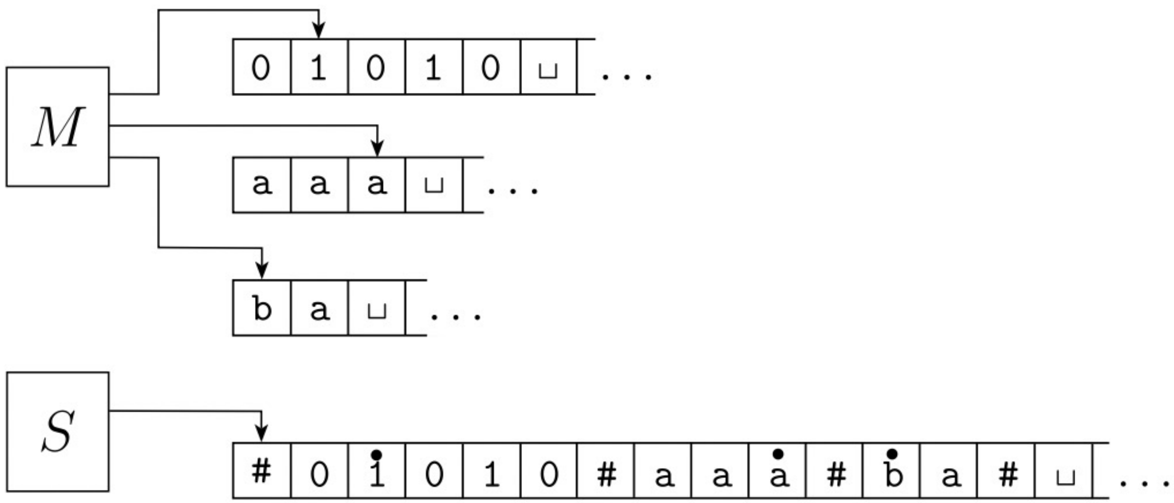
To this aim, we use a new symbol # as a delimiter to separate the contents of the different tapes. instead of space

Moreover, S must keep track of the locations of the heads.

This is done by adding a new «dotted» copy of each tape symbol

→ A dotted symbol marks the place where the head on that tape is. these are added to Gamma

EXAMPLE (with $k = 3$):



Multitape TMs vs single tape TMs

On input $w = w_1 \ldots w_n$:

1. First S puts its tape into the format that represents all k tapes of M:

$$\#\dot{w}_1 w_2 \cdots w_n \#\dot{\sqcup}\#\dot{\sqcup}\# \cdots \#$$

2. To simulate a single move:
   - First S scans its tape from the first # (which marks the left-hand end) to the $(k+1)^{st}$ # (which marks the right-hand end) in order to determine the symbols under the virtual heads
   - Then S makes a second pass to update the tapes according to the way that M's transition function dictates

3. If at any point S moves one of the virtual heads to the right on a #, this means that M has moved the corresponding head on the blank portion of that tape.

   → S shifts one position to the right all the following tape symbols (including that #) and overwrites the # with a blank symbol on that cell

we have to make space for new eventual writing

Q.E.D.

Variants of TMs: Nondeterminism

A nondeterministic TM is defined by having, at any point in a computation, several possibilities.

This is modeled by having the transition function defined as

$$\delta : (Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\}) \times \Gamma \rightarrow P(Q \times \Gamma \times \{L,R\})$$

The computation of a NTM is a tree whose branches correspond to different possibilities;

if some branch leads to the accept state, the machine accepts its input. we need AT LEAST one

trivially, non-acceptance does not mean rejecting as TM can go on forever

Thm.: Every nondeterministic TM N has an equivalent deterministic TM D.

Proof

We view N's computation on an input w as a tree.

Each branch of the tree represents one of the branches of the nondeterminism.

Each node of the tree is a configuration of N, with the root being the starting configuration.

D visits this tree, looking for an accepting configuration by using a breadth-first search

→ This strategy explores all nodes at the same depth before exploring nodes at the next depth

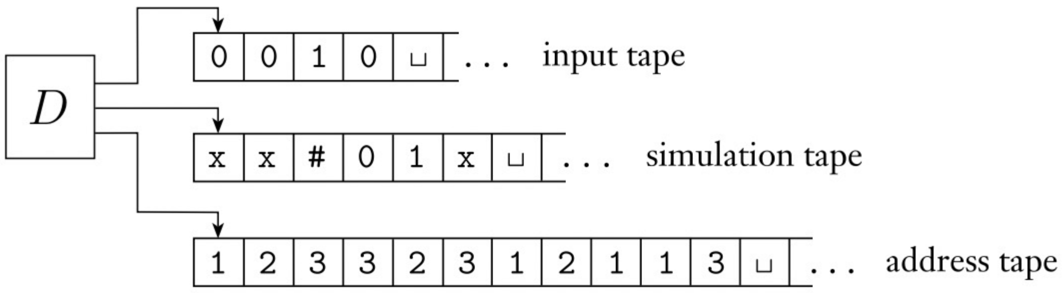
REMARK: using a depth-first search could lead to visiting one infinite branch (non-termination!) and miss an accepting configuration on some other branch

Equivalence of NTMs and (D)TMs

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

D has three tapes (this does not add power to TMs):

1. Tape 1 always contains the input string and is never altered
2. Tape 2 maintains a copy of N's tape on some branch of its nondeterministic computation
3. Tape 3 keeps track of D's location in N's nondeterministic computation tree.



Encoding the address of a node:

- Every node can have at most b children, where b is maximum number of possible choices in N's transition function → let's fix an order among all possibilities for all states
- Every node in the tree has an address that is a string over the alphabet $\{1, 2, \ldots, b\}$:
  - The empty string is the address of the root
  - The i^{th} child of a node with address $x_1 \ldots x_k$ (for $x_1,\ldots,x_k \in \{1,..,b\}$ and $k \geq 0$) has address $x_1 \ldots x_k\, i$
- Sometimes a symbol may not correspond to any choice if too few choices are available for a configuration. In that case, the address is invalid and doesn't correspond to any node.

this scales to every root and every node

Equivalence of NTMs and (D)TMs

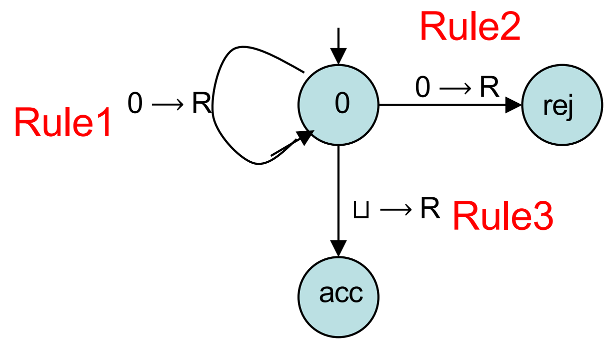
D with input *w* behaves as follows:

1. Initially, tape 1 contains the input *w*; tapes 2 and 3 are empty.
2. Copy tape 1 to tape 2
3. Use tape 2 to simulate *N* with input *w* on the branch denoted by the content of tape 3:
   - Scan all symbols on tape 3 to determine the choice to make (among those allowed by *N*'s transition function) and and correspondingly evolve tape 2
   - If the address is invalid, go to 4 (i.e., abort this run)
   - If an accepting configuration is encountered, ACCEPT the input.
4. Replace the string on tape 3 with the next string in the string ordering and go to 2.

Q.E.D.

An example

Consider the following NTM:



With input 00 its computation tree is:

ε = q_0 00

1 = 0 q_0 0 0 q_{rej} 0 = 2

str is done here only choice is blank

11 = 00 q_0 00 q_{rej} = 12

113 = 00 q_{acc}

So, the associated DTM explores the sequences:

- Lev. 0: ε
- Lev. 1: 1, 2, 3 (invalid)
- Lev. 2: 11, 12, 13 (invalid), 21 (inv.), 22 (inv.), 23 (inv.), 31 (inv.), 32 (inv.), 33 (inv.)
- Lev. 3: 111 (invalid), 112 (invalid), 113 ACCEPT