Computational Models — Lecture 7

Handout Mode

Iftach Haitner.

Tel Aviv University.

December 04, 2017

---

1 Based on frames by Benny Chor, Tel Aviv University, modifying frames by Maurice Herlihy, Brown University.
Talk Outline

▶ Turing Machines (TMs)

▶ Multitape TMs, RAMs

▶ Church-Turing Thesis

▶ Non-deterministic TM (if time permits)

● Sipser’s book, 3.1 , 3.2, & 3.3
A Finite Automaton

01101 1001

read  unread
A Pushdown Automaton

01101 1001

read unread

a b a

pop

push
A Turing Machine
Part I

Turing Machines (TMes)
Turing Machines

- Machines so far (DFA, PDA) read input only once

Turing Machines

- Can go back and forth over the input
- Can overwrite the input
- Can write information on tape and come back to it later

Input string is written on a tape:

- At each step, machine reads a symbol, and then
  - writes a new symbol, and
  - moves read/write head to either right or left.
  - changes its state
TM vs. PDA vs. DFA

- A Turing machine can both write on the tape and read from it.
- A PDA is restricted to reading from the stack in LIFO manner.
- A DFA has no media for writing anything – it must all be in its finite state.
- The TM read-write head can move both to the left and to the right.
- The TM read-write tape is infinite to the right.
- The special final (accepting/rejecting) states of TM take immediate effect (so the head need not be at some special position).
Effects of a single step

- Changes current state
- Changes head position and tape content at current position.

- Each step has very local, small effect.
- Yet, many small effects can accumulate to a meaningful task.
Example $B = \{ w\#w : w \in \{0, 1\}^* \}$

<table>
<thead>
<tr>
<th>Algorithm 1 (A TM deciding $B$ (pseudocode))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Check for a single $#$,</td>
</tr>
<tr>
<td>▶ If false, reject.</td>
</tr>
<tr>
<td>2. Zig-zag across the tape, check identical letters, and replace them by $X$.</td>
</tr>
<tr>
<td>▶ If not identical, reject.</td>
</tr>
<tr>
<td>3. When all the letters left of $#$ are marked $X$, check for remaining letters right of $#$</td>
</tr>
<tr>
<td>▶ If there are remaining letters, reject.</td>
</tr>
<tr>
<td>▶ Otherwise, accept</td>
</tr>
</tbody>
</table>
Example $B = \{ w \# w : w \in \{0, 1\}^* \}$ cont.

- Input: $0 1 1 0 0 0 \# 0 1 1 0 0 0$
- $\overline{0} 1 1 0 0 0 \# 0 1 1 0 0 0$
- $X \overline{1} 1 0 0 0 \# 0 1 1 0 0 0$

... 

- $X 1 1 0 0 0 \# \overline{0} 1 1 0 0 0$
- $X 1 1 0 0 0 \# X 1 1 0 0 0$

... 

- $\overline{X} 1 1 0 0 0 \# X 1 1 0 0 0$
- $X \overline{1} 1 0 0 0 \# X 1 1 0 0 0$
- $X X \overline{1} 0 0 0 \# X 1 1 0 0 0$

... 

- $X X X X X X \# X X X X X X$
### Definition 2 (Turing machine)

A Turing machine (TM) is a 7-tuple \((Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)\), where

- \(Q\) is a finite set of states.
- \(\Sigma\) the input alphabet not containing the blank symbol \(\bot\).
- \(\Gamma\) is the tape alphabet, where \(\bot \in \Gamma\) and \(\Sigma \subset \Gamma\).
- \(\delta : Q \setminus \{q_a, q_r\} \times \Gamma \ightarrow Q \times \Gamma \times \{L, R\}\) is the transition function.
- \(q_0 \in Q\) is the start state.
- \(q_a \in Q\) is the accept state.
- \(q_r \in Q \setminus \{q_a\}\) is the reject state.
**Transition Function** \( \delta \)

\[
\delta : Q \setminus \{q_a, q_r\} \times \Gamma \mapsto Q \times \Gamma \times \{L, R\}
\]

Informally, \( \delta(q, a) = (r, b, L) \) “means":
in state \( q \) where head reads tape symbol \( a \), the machine:

- **Writes** \( b \) over \( a \) (\( a = b \) is possible),
- **Enters** state \( r \),
- **Moves** the head left (this is what the \( L \) stands for).
Model of Computation (informal)

Before we start:

- Input $w = w_1w_2\ldots w_n \in \Sigma^*$ is placed on $n$ leftmost tape cells, one tape cell per input letter.
  Rest of tape contains blanks $\square$

- since $\square \notin \Sigma$, leftmost blank indicates end of input.

- read/write head is on leftmost cell of tape

The “computation”:
$M$ “computes” according to transition function $\delta$.
Computation continues until $q_a$ or $q_r$ is reached. Otherwise $M$ runs forever.

- If $M$ tries to move head beyond left-hand-end of tape, it doesn’t move (still $M$ does not crash)
Configurations

A TM configuration is a convenient notation for recording the state, head location, and tape contents of a TM in a given instant. Think of it as a snapshot.

For example, configuration $1011q_70111$ means:

- Current state is $q_7$
- Left hand side of tape (to the left of the head) is $1011$
- Right hand side of tape is $0111$ followed by infinite number of $\_\_\_'s$
- Head is on $0$ (leftmost entry of right hand side).
  (i.e., head location is fifth cell)

Configuration is a finite string in $\Gamma^*Q\Gamma^*$.

Given a configuration $C$, let $\text{head}(C)$ be the location of the “head” in $C$. e.g., $\text{head}(1011q_70111) = 5$. 
Special configurations

- Starting configuration (with respect to a word $w$): $q_0 w$
- Accepting configuration: $w' q_a w'$
- Rejecting configuration: $w' q_r w''$
- Halting configurations: Accepting and Rejecting configurations.
The yield relation

Configuration \( C = (x' q x'') \) yields \( D = (y' p y'') \) with respect to TM \( M = (\ldots, \delta, \ldots) \), if the transition from \( C \) to \( D \) is justified by \( \delta \):

Let \( x = xx'' \) and \( y = y' y'' \), and let \( (q', d, X) = \delta(q, x_{\text{head}}(C)) \).

1. \( p = q' \).
2. \( y_{\text{head}}(C) = d \).
3. \( \text{head}(D) = \text{head}(C) - 1 \) if \( X = L \), and \( \text{head}(C) + 1 \) o/w (i.e., \( X = R \)).
4. \( |C| = |D| \) (hence, \( |x| = |y| \))
5. \( (x)_j = (y)_j \) for all \( j \in \{1, \ldots, |x|\} \setminus \{\text{head}(C)\} \).

(no other changes in content)

Example: if \( \delta(q, c) = (p, d, L) \), then \( uaqcv \) yields \( upadv \).

Special cases:

1. If \( \text{head}(C) = 1 \) (i.e., head is at left end) and \( \delta(q, x_{\text{head}}(C)) = (\cdot, \cdot, L) \), then item 2 changes to \( \text{head}(D) = \text{head}(C) \).
2. \( C = wq \) implies \( D \), if \( wq_\perp \) implies \( D \) (i.e., , new configuration is longer, as it “annexed” one blank.)
Accepting and rejecting a word

A sequence of configurations $C_1, C_2, \ldots, C_k$ is valid, with respect to TM $M$ and $w \in \Sigma^*$, if

1. $C_1$ is the start configuration of $M$ on $w$ (i.e., $C_1 = q_0 w$)
2. Each $C_i$ yields $C_{i+1}$

A pair of TM $M$ and input $w \in \Sigma^*$, induces a single, (possibly) infinite sequence of configurations $C_1, C_2, C_3 \ldots$, that any prefix of it is a valid sequence, and any valid sequence is its prefix.

A sequence of configurations $C_1, \ldots, C_k$ is accepting, if $C_k$ is an accepting configuration.

A sequence of configurations $C_1, \ldots, C_k$ is rejecting, if $C_k$ is a rejecting configuration.

TM $M$ accepts an input $w \in \Sigma^*$, if exists a valid with respect to $M$ and $w$, accepting sequence of configurations.

TM $M$ rejects an input $w \in \Sigma^*$, if exists a valid, with respect to $M$ and $w$, rejecting sequence of configurations.
Enumerable and decidable languages

The strings in $\Sigma^*$ accepted by a TM $M$, denoted $L(M)$, is the language of $M$.

**Definition 3**

A language is recursively enumerable, $\mathcal{RE}$, if some TM accepts it. (In the book called Turing-recognizable)

On given input, a TM may

- **halt** — either accept or reject
- **loop** — not halt (run forever)

**Major concern:** In general, we never know if the TM will halt.

**Definition 4**

A TM $M$ decides a language, if it accepts it, and halts on every word in $\Sigma^*$.

Such TM is called a decider.

**Definition 5**

A language is decidable, $\mathcal{R}$, if some TM decides it. (In the book called Turing-decidable, also known as, recursive)
Part II

Examples
Example $A = \{0^{2^n} : n \geq 0\}$

Algorithm 6 (A TM deciding $A$ (pseudocode))

On input $w$:

1. Reject, if tape is empty.
2. Accept, if tape contains a single 0.
3. Reject, if tape contains odd number of 0.
4. Move to the right, "erasing" every other 0
5. Return to the start of the tape, and go to Step 2

Correctness?
Example $A = \{0^{2^n} : n \geq 0\} – \text{TM formal definition}$

**Definition 7 ($M_2 = (Q, \Sigma, \Gamma, \delta, q_1, q_a, q_r)$)**

- $Q = \{q_1, q_2, q_3, q_4, q_5, q_a, q_r\}$
- $\Sigma = \{0\}$ (input alphabet)
- $\Gamma = \{0, x, \omega\}$ (tape alphabet)
- $q_1$ start state
- $q_a$ accept state
- $q_r$ reject state
Diagram conventions

- $\delta(q, a) = (p, X)$ stands for $\delta(q, a) = (p, a, X)$. (input symbol unchanged)
- If $\delta(q, a)$ is undefined, it means $\delta(q, a) = (q_r, \cdot, \cdot)$

- $\begin{array}{c}
q \\
\xrightarrow{a \rightarrow b, X}
\end{array} p$ stands for $\delta(q, a) = (p, b, X)$

- $\begin{array}{c}
q \\
\xrightarrow{a \rightarrow X}
p
\end{array}$ stands for $\delta(q, a) = (p, a, X)$
The TM “marks” the leftmost cell with $\downarrow$.
**Example** $B = \{ w \# w : w \in \{0, 1\}^* \} – TM$ formal definition

### Definition 8 ($M_1 = (Q, \Sigma, \Gamma, \delta, q_1, q_a, q_r)$)

- $Q = \{ q_1, \ldots, q_{14}, q_a, q_r \}$.  
- $\Sigma = \{0, 1, \#\}$ (input alphabet)
- $\Gamma = \{0, 1, \#, x, \omega\}$ (tape alphabet)
- $q_1$ start state
- $q_a$ accept state
- $q_r$ reject state
Example $B = \{w \# w : w \in \{0, 1\}^*\}$ – Transition function
Example: \[ C = \{a^i b^j c^k : i \cdot j = k \land i, j, k \geq 1\} \]

**Algorithm 9 (A TM deciding \( C \) (pseudocode))**

1. Scan from left to right to check that input is \( a^+ b^+ c^+ \)
2. Return to start of tape
3. Cross off one \( a \) and scan right until \( b \) occurs.
   Shuttle between \( b \)'s and \( c \)'s, crossing off one of each, until all \( b \)'s are gone.
4. Restore the crossed-off \( b \)'s and repeat previous step if more \( a \)'s exist.
   If all \( a \)'s crossed off, check if all \( c \)'s crossed off.
   If yes, accept; otherwise reject.
**Example: element distinctness**

Consider the *element distinctness* problem

\[ E = \{ \#x_1\#x_2\# \ldots \#x_\ell : x_i \in \{0, 1\}^* \land i \neq j \implies x_i \neq x_j \} \]

Verbally,

- List of strings in \( \{0, 1\}^* \) separated by \('#'s.
- List is in language (\& machine *should* accept), if all strings are different.
Decider for $E = \{ \# x_1 \# x_2 \# \ldots \# x_\ell : x_i \in \{0,1\}^* \land i \neq j \implies x_i \neq x_j \}$

Algorithm 10 (A TM deciding $E$ (pseudocode))

Input: $w$

1. Place a “mark” on leftmost tape symbol. If symbol not $\#$, Reject.
2. Scan right to next $\#$ and place mark on top. If none, Accept.
3. Compare the two strings to right of marked $\#$’s (how?). If equal, reject.
4. If possible, move rightmost mark to next $\#$ on right and go to Step 3.
5. If possible, move leftmost mark to next $\#$ on right and rightmost mark to $\#$ after that, and go to Step 3.
6. Accept.
Decider for \( E = \{ \#x_1\#x_2\# \ldots \#x_\ell : x_i \in \{0, 1\}^* \land i \neq j \implies x_i \neq x_j \} \)

Question 11

How do we “mark” a symbol?

Answer: For each tape symbol \#, add tape symbol \# to the tape alphabet \( \Gamma \).
Part III

Equivalents Variants
Turing machine variants

For example, suppose the Turing machine head is allowed to stay put.

\[ \delta : Q \setminus \{q_a, q_r\} \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\} \]

**Question 12**

Does this add any power?

**Answer:** No. Replace each S transition with two transitions: R then L. (ahmm ... why not vice-versa?)

**Question 13**

Does this reduce power?

Machine \( M \) emulates machine \( N \), if \( M \) accepts the same inputs and halts on the same inputs as \( N \) does.

Computational models \( A \) and \( B \) are equivalent, if there is two-way emulation: every machine in model \( A \) has a machine in model \( B \) that emulates it, and vice versa.
Multitape Turing Machines

- Constant number of infinite tapes
- Each tape has its own head
- Initially, input string is placed on tape 1 and rest tape are blank

Transition function is of the form

\[ \delta : Q \setminus \{q_a, q_r\} \times \Gamma^k \mapsto Q \times \Gamma^k \times \{L, R\}^k \]

The expression \( \delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, L, R, \ldots, L) \) “means”: assuming the machine is in state \( q_i \) and heads 1 through \( k \) reading \( a_1, \ldots, a_k \):

- the machine goes to state \( q_j \),
- heads 1 thought \( k \) write \( b_1, \ldots, b_k \),
- each head moves right or left as specified.
**$k$-tape TM’s, configurations**

1. Configurations are of the form $(\Gamma^* Q \Gamma^*)^k$

   Configuration $w'_1 q w''_1 \$ w'_2 q w''_2 \$ \ldots \$ w'_k q w''_k \$ "$means":
   
   1.1 State is $q$
   1.2 Content of $i$’th tape is $w'_i w''_i$
   1.3 Head location of $i$’th tape is $|w'_i| + 1$

2. Starting configuration (for input $w$) is $q w \$ q \$ \ldots \$ q$.

3. The yield relation is a natural extension of the single tape (see next slide)

4. Accepting and rejecting is defined analogously to the single tape.
Multitape TM’s, yield relation

Configuration \( C = (C_1 = (x'_1 q x''_1) \ldots C_k = (x'_k q x''_k)) \) yields configuration \( D = (D_1 = (y'_1 p y''_1) \ldots D_k = (y'_k p y''_k)) \), if:

Let \((q', (d_1, \ldots, d_k), (X_1, \ldots, X_K)) = \delta(q, (x''_1)_1, \ldots, (x''_k)_1)\). Then

1. \( p = q' \)

2. \( \forall i \in [k]: \)

   2.1 \( \text{head}(D_i) = \text{head}(C_i) - 1 \) if \( X_i = L \), and \( \text{Head}(C + i) - 1 \) o/w.

   2.2 \( (y'_i y''_i)_{\text{head}(C_i)} = d_i. \)

   2.3 \( |C_i| = |D_i| \)

   2.4 \( (y'_i y''_i)_j = (x'_i x''_i)_j \) for all \( j \in \{1, \ldots, |x_i|\} \setminus \{\text{head}(C_i)\} \).

Special cases:

::
Equivalence of multitape TM and singletape ones

It is clear that a multitape TM can emulate a singletape TM.

**Theorem 14**

*For any multitape TM there exists a singletape TM that emulates it.*

**Corollary 15**

*A language is enumerable [resp., decidable], if and only if there is some multitape Turing machine that accepts [resp., decides] it.*

Proof: We will show how to “convert” a multitape TM, $M$, into an equivalent singletape TM, $S$. 
Emulation idea

▶ S emulates $k$ tapes of $M$ by storing them all on a single tape with delimiter #.

▶ S marks the current positions of the $k$ heads by placing • “above” the letters in current positions. It “knows” which tape the mark belongs to by counting (up to $k$) from the #’s to the left.
The emulator

Algorithm 16 (The emulator (pseudocode))

On input \( w = w_1 \cdots w_n \):

1. Write on the tape \( \# \ w_1 \ w_2 \cdot w_n \# \ \# \ \# \ \# \ \# \ \cdots \ \# \)

2. Scan the tape from first \( \# \) to \((k + 1)\)-st \( \# \) to read symbols under “virtual” heads.

3. Rescan the tape to write new symbols and move heads

4. When \( M \) moves head onto unused blank square, \( S \) will try to move virtual head onto \( \# \).
   \( S \) handles it by writing blank \( \square \) on tape, and shifting the rest of tape one square to the right.
Random Access Machine (RAM)

- CPU
- 3 Registers (Instruction Register (IR), Program Counter (PC), Accumulator (ACC))
- Memory
- Operation:
  - Increment PC
  - Set IR ← MEM[PC]
  - Execute instruction in IR
  - Repeat
- Instructions are typically compare, add/subtract, multiply/divide, shift left/right.
- We assume no limit on the registers’ size
- All instructions are doable on a TM.
RAM: schematic picture
**RAM: instructions**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 HALT</td>
<td>Stop Computation</td>
</tr>
<tr>
<td>01 LOAD x</td>
<td>ACC ← MEM[x]</td>
</tr>
<tr>
<td>02 LOADI x</td>
<td>ACC ← x</td>
</tr>
<tr>
<td>03 STORE x</td>
<td>MEM[x] ← ACC</td>
</tr>
<tr>
<td>04 ADD x</td>
<td>ACC ← ACC + MEM[x]</td>
</tr>
<tr>
<td>05 ADDI x</td>
<td>ACC ← ACC + x</td>
</tr>
<tr>
<td>06 SUB x</td>
<td>ACC ← ACC - MEM[x]</td>
</tr>
<tr>
<td>07 SUBI x</td>
<td>ACC ← ACC - x</td>
</tr>
<tr>
<td>08 JUMP x</td>
<td>PC ← x</td>
</tr>
<tr>
<td>09 JZERO x</td>
<td>PC ← x if ACC = 0</td>
</tr>
<tr>
<td>10 JGT x</td>
<td>PC ← x if ACC &gt; 0</td>
</tr>
</tbody>
</table>
RAM: example program

A program that multiplies two numbers (in locations 1000 & 1001), and stores the result in 1002

<table>
<thead>
<tr>
<th>Memory</th>
<th>Machine Code</th>
<th>Assembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>011000</td>
<td>LOAD 1000</td>
</tr>
<tr>
<td>0002</td>
<td>031003</td>
<td>STORE 1003</td>
</tr>
<tr>
<td>0003</td>
<td>020000</td>
<td>LOADI 0</td>
</tr>
<tr>
<td>0004</td>
<td>031002</td>
<td>STORE 1002</td>
</tr>
<tr>
<td>0005</td>
<td>021003</td>
<td>LOAD 1003</td>
</tr>
<tr>
<td>0006</td>
<td>090013</td>
<td>JZERO 0013</td>
</tr>
<tr>
<td>0007</td>
<td>070001</td>
<td>SUBI 1</td>
</tr>
<tr>
<td>0008</td>
<td>031003</td>
<td>STORE 1003</td>
</tr>
<tr>
<td>0009</td>
<td>011002</td>
<td>LOAD 1002</td>
</tr>
<tr>
<td>0010</td>
<td>041001</td>
<td>ADD 1001</td>
</tr>
<tr>
<td>0011</td>
<td>080004</td>
<td>JUMP 4</td>
</tr>
<tr>
<td>0013</td>
<td>000000</td>
<td>HALT</td>
</tr>
</tbody>
</table>
Theorem 17

A Turing machine can emulate this RAM model.

Proof’s idea: We can emulate the RAM model with a multi-tape Turing machine:

- One tape for each register (IR, PC, ACC)
- One tape for the Memory
- Memory tape will be entries of the form `<address>` `<contents>` in increasing order of addresses.
**Emulator’s Tapes**

**Memory**

<table>
<thead>
<tr>
<th>0001</th>
<th>011000 &amp; 0002</th>
<th>031003 &amp; 0003</th>
<th>020000 &amp; 0004</th>
<th>031002</th>
</tr>
</thead>
</table>

- **address**
- **contents**

**Instruction Pointer**

| 0001 |

| ... |

**Instruction Register**

| 11000 |

| ... |

**Accumulator**

| 0 |

| ... |
Algorithm 18 (Emulator (high-level description))

1. Scan through memory until reach an address that matches the PC
2. Copy contents of memory at that address to the IR
3. Increment PC
4. Based on the instruction code:
   - Copy a value into PC
   - Copy a value into Memory
   - Copy a value into the ACC
   - Perform an arithmetic operation, a shift, or a comparison
Part IV

Turing Completeness
Turing completeness

- A computation model is called **Turing complete**, if it can emulate a Turing Machine.
- Turing complete machine can compute anything a TM could
  - Of course it might *not* be convenient ...
- We just seen (the easy part of the proofs) that multitape TM, and RAM machines are Turing complete
Part V

Church-Turing Thesis
Beyond RAM

- A RAM can be modeled (emulated) by a Turing Machine.
- Any current machine (architecture, manufacturer, operating system, power supply, etc.) can be modeled by a Turing Machine.
- Note that at this point, we don’t care how long it might take, just that it can be done.
- Hence, if there is an “algorithm" for it, a Turning Machine can do it.
What is an algorithm?

Informally:

1. A recipe

2. A procedure

3. A computer program

4. Who cares? I know it when I see it :-(

5. The notion of algorithm has long history in Mathematics (starting with Euclid’s gcd algorithm), but not precisely defined until 20’th century

   Informal notions rarely questioned, still they were insufficient
Computation models

- Many models have been proposed for general-purpose computation. Remarkably, all “reasonable” models were shown to be equivalent to Turing machines.
- The notion of an algorithm is model-independent!
- We don’t really care about Turing machines per se.
- We do care about understanding computation, and because of their simplicity, Turing machines are a good model to use.
Models equivalent to TM

- All “reasonable” programming languages (e.g., Java, Pascal, C, Python, Scheme, Mathematica, Maple, Cobol, ...).
- $\lambda$-calculus of Alonzo Church
- Turing machines of Alan Turing
- Recursive functions of Godel and Kleene
- Counter machines of Minsky
- Normal algorithms of Markov
- Unrestricted grammars
- Two stack automata
- Random access machines (RAMs)
Church-Turing Thesis

“The intuitive notion of reasonable models of computation equals Turing machine algorithms”.

[Images of Alan Turing and Alonzo Church]
Consider MUntel’s $\aleph$-AXP10© processor (to be released ...)

**Definition 19 ($\aleph$-AXP10©)**

Like a Turing machine, except

- Takes first step in 1 second.
- Takes second step in $1/2$ second.
- Takes $i$-th step in $2^{-i}$ seconds ...

After 2 seconds, the $\aleph$-AXP© decides any enumerable language!

**Question 20**

Does the $\aleph$-AXP© invalidate the Church-Turing Thesis?
Part VI

Non-deterministic Turing machines (NTMs)
**Non-Deterministic Turing Machines (NTMs)**

NTM \( N = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r) \),

\[
\delta : Q \times \Gamma \mapsto \mathcal{P}(Q \times \Gamma \times \{L, R\})
\]

The yield relation (for NTM): configuration \( C \) yields \( D \), if it yields it, according to the deterministic definition, for some deterministic restriction of \( \delta \).

Configuration \( C = (x'qx'') \) yields \( D = (y'py'') \), if 

\((p, (y'y'')_{\text{head}(C)}, X) \in \delta(q, (x'x'')_{\text{head}(C)}) \) and . . .

The computation tree of \( N \) on input \( w \):

- **Root** is the starting configuration (with respect to \( w \))
- **The children** of a node are all configurations it (directly) yields.
  
  # of children is at most: \(|Q| \cdot |\Gamma| \cdot 2\).

Valid sequences of configurations with respect to \( N \) and \( w \), are defined as in the deterministic case.

Any rooted finite path in the computation tree of \( N \) and \( w \), it is a valid sequence of configurations (with respect to \( N \) and \( w \)).
Accepting a word

$N$ accepts $w \in \Sigma^*$, if $\exists$ an accepting path (i.e., accepting sequence of configurations) in its computation tree of $N$ on $w$.

(Equivalently, $\exists$ an accepting sequence of configurations, with respect to $N$ and $w$.)

$N$ halts on $w \in \Sigma^*$, if it accepts it, or the computation tree of $N$ on $w$ is finite: $\exists k \in \mathbb{N}$, such that there is no valid sequence of length $k$. 
Equivalence of TM and NTM

It is clear that NTM is Turing complete.

**Theorem 21**

*For any non-deterministic TM there exists a deterministic TM that emulates it.*

**Corollary 22**

*A language is enumerable [resp., decidable], if and only if there is some non-deterministic Turing machine that accepts [resp., decides] it.*

We will prove a slightly simpler to prove result.

**Theorem 23**

*For any NTM $N$ there exists TM $D$ with $L(N) = L(D)$.*
Basic idea

- $D$ tries all possible branches in $N$ computation tree
- If $D$ finds any accepting path, it accepts.

Question 24

How to traverse this tree?
- depth-first search?
- breadth-first search?
The machine $D$ has three tapes:

- **Input** tape is never altered (only read from),
- **Emulation** tape serves as $N$’s tape,
- **Address** tape keeps track of $D$’s location in $N$’s *computation tree*. 
Address tape

Let $b$ be bound on the number of children of node in $N$’s computation tree.

The address tape contains a pointer into the configuration tree. Concretely, a string in $\Sigma^*_b$, for $\Sigma_b = \{1, \ldots, b\}$.

Definition 25

By incrementing the value of the address tape, we mean replace its content with the next string in $\Sigma^*_b$, according to the lexicographic order.

Example (for $b = 2$): $\varepsilon \mapsto 1 \mapsto 2 \mapsto 11 \mapsto 12 \mapsto 21 \mapsto 22 \mapsto 111 \ldots$

Question 26

Can a TM implement the increment function?

Question 27

Can a (deterministic) TM compute the value of the node indexed by the address tape (with respect to TM $N$ and input $w$)?
Algorithm 28 (TM $D$ (pseudocode))

1. Compute the configuration of $N$ indexed by the address tape:
   1.1 Copy input tape (i.e., $w$) to emulation tape.
   1.2 Use emulation tape to emulate the run of $N$ on $w$, using the address tape to resolve non-deterministic choices.
      Break current emulation, if
      ★ End of path (i.e., symbols on address tape are exhausted)
      ★ Accepting/Rejecting configuration is reached
      ★ Non-deterministic choice is invalid

2. Accept, if an accepting configuration was reached.

3. Increment the value of address tape.

4. Go back to Step 1.

Question 29

Change $D$ to emulate $N$. 