1. Basic Definitions

- Turing machines are used as the formal model of algorithms.
- Turing machines can simulate arbitrary algorithms with inconsequential loss of efficiency using a single data structure: a string of symbols.

**Definition.** A Turing machine is a quadruple $M = (K, \Sigma, \delta, s)$ with a finite set of **states** $K$, a finite set of symbols $\Sigma$ (*alphabet* of $M$) so that $\sqcup, \triangleright \in \Sigma$, a transition function $\delta$: $K \times \Sigma \rightarrow (K \cup \{h, "yes", "no"\}) \times \Sigma \times \{\rightarrow, \leftarrow, \cdot\}$, a halting state $h$, an accepting state “yes”, a rejecting state “no”, and cursor directions: $\rightarrow$ (right), $\leftarrow$ (left), and $\cdot$ (stay).

### Transition functions

- Function $\delta$ is the “program” of the machine.
- For the current state $q \in K$ and the current symbol $\sigma \in \Sigma$,
  - $\delta(q, \sigma) = (p, \rho, D)$ where $p$ is the new state,
  - $\rho$ is the symbol to be overwritten on $\sigma$, and
  - $D \in \{\rightarrow, \leftarrow, \cdot\}$ is the direction in which the cursor will move.
- For any states $p$ and $q$, $\delta(q, \triangleright) = (p, \rho, D)$ with $\rho = \triangleright$ and $D = \rightarrow$.
- If the machine moves off the right end of the string, it reads $\sqcup$ (the string becomes longer but it cannot become shorter; thus it keeps track of the space used by the machine).

**Example.** Consider a Turing machine $M = (K, \Sigma, \delta, s)$ with $K = \{s, q\}$, $\Sigma = \{0, 1, \sqcup, \triangleright\}$ and a transition function $\delta$ defined as follows:

<table>
<thead>
<tr>
<th>$p \in K$</th>
<th>$\sigma \in \Sigma$</th>
<th>$\delta(p, \sigma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s$, 0</td>
<td></td>
<td>$(s, 0, \rightarrow)$</td>
</tr>
<tr>
<td>$s$, 1</td>
<td></td>
<td>$(s, 1, \rightarrow)$</td>
</tr>
<tr>
<td>$s$, $\sqcup$</td>
<td></td>
<td>$(q, \sqcup, \leftarrow)$</td>
</tr>
<tr>
<td>$s$, $\triangleright$</td>
<td></td>
<td>$(s, \triangleright, \rightarrow)$</td>
</tr>
<tr>
<td>$q$, 0</td>
<td></td>
<td>$(h, 1, \leftarrow)$</td>
</tr>
<tr>
<td>$q$, 1</td>
<td></td>
<td>$(q, 0, \leftarrow)$</td>
</tr>
<tr>
<td>$q$, $\triangleright$</td>
<td></td>
<td>$(h, \triangleright, \rightarrow)$</td>
</tr>
</tbody>
</table>

The machine computes $n + 1$ for a natural number $n$ in binary.
The program starts with (i) initial state $s$,
(ii) the string initialized to $\triangleright x$ where $x$ is a finitely long string in $(\Sigma - \{\sqcup\})^*$ ($x$ is the input of the machine) and
(iii) the cursor pointing to $\triangleright$.

A machine has halted iff one of the 3 halting states

(h, “yes”, “no”) has been reached.

If “yes” has been reached, the machine accepts the input.
If “no” has been reached, the machine rejects the input.

Output $M(x)$ of a machine $M$ on input $x$:
(i) If $M$ accepts/rejects, then $M(x) = “yes” / “no”$.
(ii) If $h$ has been reached, $M(x) = y$ where $\triangleright y \sqcup \ldots$ is the string of $M$ at the time of halting.
(iii) If $M$ never halts on input $x$, then $M(x) = \gamma$

Operational semantics

A configuration $(q,w,u)$:
$q \in K$ is the current state and $w,u \in \Sigma^*$ where
(i) $w$ is the string to the left of the cursor including the symbol scanned by the cursor and
(ii) $u$ is the string to the right of the cursor.

The relation $\xrightarrow{M}$ (yields in one step):  $(q,w,u) \xrightarrow{M} (q',w',u')$

Let $\sigma$ be the last symbol of $w$ and $\delta(q,\sigma) = (p,\rho,D)$.
Then $q' = p$, and $w',u'$ are obtained according to $(p,\rho,D)$.

Example. If $D = \xrightarrow{}$, then
(i) $w'$ is $w$ with its last symbol replaced by $\rho$ and the first symbol of $u$
appended to it ($\sqcup$ if $u$ is empty) and
(ii) $u'$ is $u$ with the first removed (or empty, if $u$ is empty).

2. Turing Machines as Algorithms

Turing machines are natural for solving problems on strings:

Let $L \subseteq (\Sigma - \{\sqcup\})^*$ be a language.
A Turing machine $M$ decides $L$ iff for every string $x \in (\Sigma - \{\sqcup\})^*$,
if $x \in L$, $M(x) = “yes”$ and
if $x \notin L$, $M(x) = “no”$.

If $L$ is decided by a Turing machine, $L$ is a recursive language.

A Turing machine $M$ computes a (string) function
$f: (\Sigma - \{\sqcup\})^* \rightarrow \Sigma^*$ iff for every string $x \in (\Sigma - \{\sqcup\})^*$,
$M(x) = f(x)$.

If such an $M$ exists, $f$ is called a recursive function.
Example. Transition function $\delta$ for checking even parity of $x \in \{0, 1\}^*$:

<table>
<thead>
<tr>
<th>$p \in K$</th>
<th>$\sigma \in \Sigma$</th>
<th>$\delta(p, \sigma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s,$</td>
<td>$\triangleright$</td>
<td>$(s, \triangleright, \rightarrow)$</td>
</tr>
<tr>
<td>$s,$</td>
<td>$0$</td>
<td>$(s, 0, \rightarrow)$</td>
</tr>
<tr>
<td>$s,$</td>
<td>$1$</td>
<td>$(t, 1, \rightarrow)$</td>
</tr>
<tr>
<td>$s,$</td>
<td>$\bot$</td>
<td>(&quot;yes&quot;, $\bot$, $\leftarrow$)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$p \in K$</th>
<th>$\sigma \in \Sigma$</th>
<th>$\delta(p, \sigma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t,$</td>
<td>$\triangleright$</td>
<td>$(t, \triangleright, \rightarrow)$</td>
</tr>
<tr>
<td>$t,$</td>
<td>$0$</td>
<td>$(t, 0, \rightarrow)$</td>
</tr>
<tr>
<td>$t,$</td>
<td>$1$</td>
<td>$(s, 1, \rightarrow)$</td>
</tr>
<tr>
<td>$t,$</td>
<td>$\bot$</td>
<td>(&quot;no&quot;, $\bot$, $\rightarrow$)</td>
</tr>
</tbody>
</table>

The respective Turing machine $M$ decides $101 \in \{0, 1\}^*$ as follows:

\[(s, \triangleright, 101) \xrightarrow{M} (s, \triangleright, 01) \xrightarrow{M} (t, \triangleright, 0, 1) \xrightarrow{M} (t, \triangleright, 101, \epsilon) \xrightarrow{M} (s, \triangleright, 101 \bot, \epsilon) \xrightarrow{M} ("yes", \triangleright, 101 \bot, \epsilon).\]

Recursively enumerable languages

- A Turing machine $M$ accepts $L$ if for every string $x \in (\Sigma - \{\bot\})^*$, if $x \in L$, then $M(x) = \text{"yes"}$ but if $x \notin L$, $M(x) = \not\right.$.
- If $L$ is accepted by some Turing machine, $L$ is a recursively enumerable language.
- We will later encounter examples of r.e. languages.

**Proposition.** If $L$ is recursive, then it is recursively enumerable.

The terms recursive and recursively enumerable suggest that Turing machines are equivalent in power with arbitrarily general (recursive) computer programs.

How does representation affect solvability?

- Any "finite" mathematical object can be represented by a finite string over an appropriate alphabet.

**Example.**

Graph:

```
1 -- 2
\|--\--
3    4
```

Representations as a string:

```
"\{(1,10),(1,11),(10,100)\}"
"(0110,0001,0000,0000)"
```
Representation vs. solvability?

- All acceptable encodings are related polynomially:
  - If A and B are both “reasonable” representations of the same set of instances, and representation A of an instance is a string with \( n \) symbols, the representation B of the same instance has length at most \( p(n) \) for some polynomial \( p \).
  - Exception: unary representation of numbers requires exponentially more symbols than the binary representation.
  - A reasonably succinct input representation is assumed. In particular, numbers are always represented in binary.

Generalized transitions

- Transitions are determined by
  \[ \delta(q, \sigma_1, \ldots, \sigma_k) = (p, \rho_1, D_1, \ldots, \rho_k, D_k). \]
  - If \( M \) is in the state \( q \), the cursor of the first string is scanning \( \sigma_1 \), that of the second \( \sigma_2 \) and so on, then the next state is \( p \), the first cursor will write \( \rho_1 \) and move \( D_1 \) and so on.

3. Turing Machines with Multiple Strings

- Turing machines with multiple strings and associated cursors are more convenient from the programmer’s point of view.
  - They can be simulated by an ordinary Turing machine with an inconsequential loss of efficiency.
  - A \( k \)-string Turing machine with an integer parameter \( k \geq 1 \) is a quadruple \( M = (K, \Sigma, \delta, s) \) where the transition function \( \delta \) has been generalized to handle \( k \) strings simultaneously:
    \[ \delta: K \times \Sigma^k \rightarrow (K \cup \{ h, "yes", "no" \}) \times (\Sigma \times \{ \leftarrow, \rightarrow, \left, \right \})^k \]
  - This definition yields an ordinary Turing machine when \( k = 1 \).

Output is defined as for ordinary machines:

- If \( (s, \leftarrow, x, \rightarrow, \ldots, \rightarrow, \epsilon) M^x \rightarrow ("yes", w_1, \ldots, w_k, u_k) \), then \( M(x) = "yes" \).
- If \( (s, \leftarrow, x, \rightarrow, \ldots, \rightarrow, \epsilon) M^x \rightarrow ("no", w_1, \ldots, w_k, u_k) \), then \( M(x) = "no" \).
- If \( (s, \leftarrow, x, \rightarrow, \ldots, \rightarrow, \epsilon) M^x \rightarrow (h, w_1, \ldots, w_k, u_k) \), then \( M(x) = y \) where \( y \) is \( w_k u_k \) with the leading \( \leftarrow \) and trailing \( \epsilon \) is removed.

The time required by \( M \) on input \( x \) is \( t \) iff

- \( (s, \leftarrow, x, \rightarrow, \ldots, \rightarrow, \epsilon) M^f \rightarrow (H, w_1, \ldots, w_k, u_k) \) where \( H \in \{ h, "yes", "no" \} \).
- If \( M(x) = / \), then the time required is thought to be \( \infty \).
Complexity classes

- Performance measured by the amount of time (or space) required on instances of size $n$ using a function of $n$.
- Machine $M$ operates within time $f(n)$ if for any input string $x$, the time required by $M$ on $x$ is at most $f(|x|)$.
- Function $f(n)$ is a time bound for $M$.
- A complexity class $\text{TIME}(f(n))$ is a set of languages $L$ decided by a multistring Turing machine operating within time $f(n)$.
- Notice that worst-case inputs are taken into account.

Multiple strings vs. a single string

Theorem. Given any $k$-string Turing machine $M$ operating within time $f(n)$, we can construct a Turing machine $M'$ operating within time $O(f(n)^2)$ and such that for any input $x$, $M(x) = M'(x)$.

Proof sketch:
- $M'$ is based on an extended alphabet $\Sigma' = \Sigma \cup \{\triangleright', \triangleleft\}$.
- $M'$ represents a configuration of $M$ by concatenation
  $$(q, w_1, u_1, \ldots, w_k, u_k) \mapsto (q, \triangleright', w'_1 u_1 < w'_2 u_2 < \ldots < w'_k u_k < \triangleleft')$$
  where each $w'_i$ is $w_i$ with the leading $\triangleright$ replaced by $\triangleright'$ and the last symbol $\sigma_i$ by $\triangleleft'$ to keep track of cursor positions.
- Initial configuration: $(s, \triangleright, \triangleright' x < \triangleleft' < \ldots < \triangleleft' < \triangleleft)$

4. Linear Speedup

- The simulation of a step of $M$ by $M'$ takes place as follows:
  1. pass: symbols underlined (scanned) on the $k$ strings
  2. pass: change in the underlined (scanned) symbols
- The strings of $M$ have a total length of $O(kf(n))$.
  To simulate one step of $M$, $M'$ needs $O(k^2 f(n))$ steps.
- Since $M$ makes at most $f(n)$ steps, $M'$ makes $O(f(n)^2)$ steps ($k$ is fixed and independent of $x$).

Thesis: No conceivable “realistic” improvement on the Turing machine will increase the domain of the language such machines decide, or will affect their speed more than polynomially.
Proof sketch

- Let $M = (K, \Sigma, \delta, s)$ be a $k$-string machine deciding $L$ in time $f(n)$. We construct a $k'$-string machine $M' = (K', \Sigma', \delta', s')$ operating within time bound $f'(n)$ and simulating $M$.
  (If $k > 1$, $k' = k$ and if $k = 1$, then $k' = 2$).
- Performance savings are obtained by adding word length: Each symbol of $M'$ encodes several symbols of $M$ and each move of $M'$ several moves of $M$.
- Given $M$ and $\varepsilon$ we take some integer $m$ and use $m$-tuples of symbols of $M$ in $M'$.
- The linear term $(n + 2)$ in the theorem is due to condensing input.

Proof sketch — cont’d

- $M'$ simulates $m$ steps of $M$ in at most a constant (6) number of steps in a stage.
- In such a stage $M'$ reads the adjacent symbols ($m$-tuples) on both sides of the cursors (this takes 4 steps).
  The state of $M'$ records all symbols at or next to all cursors.
  Now $M'$ can predict the next $m$ moves of $M$ which can be implemented in 2 steps.
- The time spent by $M'$ on input $x$ is $|x| + 2 + 6\lceil f(|x|)/m \rceil$.
- The speedup is obtained if $m = \lceil 6/\varepsilon \rceil$.
- Notice that a lot of new states have to be added: $|K| \times m^k |\Sigma|^3m^k$.

Consequences of the linear speedup theorem

- It holds for any time bound $f(n)$ such that $f(n) \geq n$,
  (i) if $f(n) = cn$, then $f'(n) \approx n$ and
  (ii) if $f(n)$ is superlinear, e.g., $f(n) = 20n^2 + 11n$, then $f'(n) \approx n^2$ (arbitrary linear speedup).
- If $L$ is polynomially decidable, then $L \in \text{TIME}(n^k)$ for some integer $k > 0$.

Definition. The set of all languages decidable by Turing machines in polynomial time $P$ is defined as the union

$$\bigcup_{k>0} \text{TIME}(n^k)$$

5. Space bounds

- Strings cannot become shorter during computation.
- Thus the sum of lengths of the final strings provides a preliminary definition of the space consumed by a computation.
- There is an overcharge: sublinear space bounds are not covered!

Example. The language of palindromes can be decided by a 3-string Turing machine in logarithmic space.
- This suggests us to exclude the effects of reading the input and writing the output as regards the consumption of space.
**Turing machines with input and output**

**Definition.** A $k$-string Turing machine ($k > 2$) with *input and output* is an ordinary $k$-string Turing machine with the following restrictions on the program $\delta$:

If $\delta(q, \sigma_1, \ldots, \sigma_k) = (p, \rho_1, D_1, \ldots, \rho_k, D_k)$, then
(a) $\rho_1 = \sigma_1$ (read-only input string),
(b) $D_k \neq \leftarrow$ (write-only output string), and
(c) if $\sigma_1 = \sqcup$, then $D_1 = \leftarrow$ (end of input respected).

**Proposition.** For any $k$-string Turing machine $M$ operating within time bound $f(n)$ there is a $(k+2)$-string Turing machine $M'$ with input and output which operates within time bound $O(f(n))$.

**Space complexity classes**

**Definition.** A space complexity class $\text{SPACE}(f(n))$ is a set of languages $L$ decidable by a Turing machine with input and output operating within space bound $f(n)$.

**Definition.** The class $\text{SPACE}(\log(n))$ is denoted by $L$.

**Example.** The language of palindromes belongs to $L$.

**Theorem.** Let $L \in \text{SPACE}(f(n))$. Then for any $\epsilon > 0$, $L \in \text{SPACE}(2 + \epsilon f(n))$.

☞ Constants do not count for space as well.

**Learning Objectives**

➤ A deeper understanding why ($k$-string) Turing machines make a reasonable model of computation.

➤ You should know how time/space complexity classes are derived using bounds on computations.

➤ The idea that multiplicative/additive constants do not count.

➤ The definitions and background of complexity classes $P$ and $L$. 

© 2005 TKK, Laboratory for Theoretical Computer Science