1. Basic Definitions

Turing machines are used as the formal model of algorithms.

Will be shown:
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 \( \sqcup, \sqcup \in \Sigma \),
a transition function \( \delta \):
\[
K \times \Sigma \rightarrow (K \cup \{h, \text{“yes”}, \text{“no”}\}) \times \Sigma \times \{\rightarrow, \leftarrow, =\},
\]
a halting state \( h \), an accepting state “yes”, a rejecting state “no”,
and cursor directions: \( \rightarrow \) (right), \( \leftarrow \) (left), and \( = \) (stay).

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

\[
\begin{array}{c|c|c}
p \in K & \sigma \in \Sigma & \delta(p, \sigma) \\
\hline
s & 0 & (s, 0, \rightarrow) \\
s & 1 & (s, 1, \rightarrow) \\
s & \sqcup & (q, \sqcup, \leftarrow) \\
s & \sqcup & (s, \sqcup, \rightarrow) \\
q & 0 & (h, 1, =) \\
q & 1 & (q, 0, \leftarrow) \\
q & \sqcup & (q, \sqcup, =) \\
q & \sqcup & (h, \sqcup, \rightarrow) \\
\end{array}
\]

The machine aims to computes \( n + 1 \) for a natural number \( n \) in binary.

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
- \( D \in \{\rightarrow, \leftarrow, =\} \) is the direction in which the cursor will move.

For any states \( p \) and \( q \), \( \delta(q, \sqcup) = (p, \rho, D) \) with \( \rho = \sqcup \) 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).
The program starts with
(i) initial state \( s \),
(ii) the string initialized to \( \triangleright x \) where \( x \) is a finitely long string in \( (\Sigma - \{\sqcup\})^\ast \) (\( x \) 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 \( M(x) \equiv \text{“yes”/“no”} \).
(ii) If \( h \) has been reached, \( M(x) = y \)
where \( \triangleright y \sqcup \triangleright \ldots \) is the string of \( M \) at the time of halting.
(iii) If \( M \) never halts on input \( x \), then \( M(x) = \) \(/

Operational semantics

A configuration \( (q, w, u) \):

\( q \in K \) is the current state and \( w, u \in \Sigma^\ast \) 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 \( M \xrightarrow{} \) (yields in one step):
\( (q, w, u) \xrightarrow{} (q', w', u') \)

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

Example. If \( D \xrightarrow{} \), then

(i) \( w' \) is \( w \) with its last symbol replaced by \( p \) 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 \( L \subset (\Sigma - \{\sqcup\})^\ast \) be a language.

A Turing machine \( M \) decides \( L \) iff for every string \( x \in (\Sigma - \{\sqcup\})^\ast \),
if \( x \in L \), \( M(x) = \text{“yes”} \) and
if \( x \notin L \), \( M(x) = \text{“no”} \).

If \( L \) is decided by a Turing machine, \( L \) is a recursive language.

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

If such an \( M \) exists, \( f \) is called a recursive function.
Example. A Turing machine \( M = (K, \Sigma, \delta, s) \) deciding the language \( L = \{ x \in \{0,1\}^* \mid \text{the number of symbols \"1\" in \( x \) is even} \} \) where \( K = \{s,t\}, \Sigma = \{0,1,\bot,\top\} \) and the transition function \( \delta \):

\[
\begin{array}{c|c|c}
 p \in K & \sigma \in \Sigma & \delta(p, \sigma) \\
\hline
 s, \top & (s, \top, \rightarrow) & t, \top \\
 s, 0 & (s, 0, \rightarrow) & t, 0 \\
 s, 1 & (t, 1, \rightarrow) & t, 1 \\
 s, \bot & ("yes", \bot, \rightarrow) & t, \bot \\
\end{array}
\]

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

\[
\begin{align*}
(s, \top, 101) \xrightarrow{\delta} (s, \top, 1, 01) \\
\xrightarrow{\delta} (t, \top, 10, 1) \\
\xrightarrow{\delta} (t, \top, 101, \varepsilon) \\
\xrightarrow{\delta} (s, \top, 101, \bot, \varepsilon) \\
\xrightarrow{\delta} ("yes", \top, 101, \bot, \varepsilon). (M(101) = \"yes\")
\end{align*}
\]

Recursively enumerable languages

\begin{itemize}
\item A Turing machine \( M \) accepts \( L \) iff for every string \( x \in (\Sigma - \{\bot\})^* \), if \( x \in L \), then \( M(x) = \text{"yes"} \) but if \( x \notin L \), \( M(x) = \text{\"no\"} \).
\item If \( L \) is accepted by some Turing machine, \( L \) is a recursively enumerable language.
\item We will later encounter examples of r.e. languages.
\end{itemize}

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.

Solving problems using Turing machines

\begin{itemize}
\item Instances of the problem need to be represented by strings.
\item Solving a decision problem amounts to deciding the language consisting of the encodings of the "yes" instances of the problem.
\item An optimization problem is solved by a Turing machine that computes the appropriate function from strings to strings (where the output is similarly represented as a string).
\end{itemize}

How does representation affect solvability?

\begin{itemize}
\item Any "finite" mathematical object can be represented by a finite string over an appropriate alphabet.
\end{itemize}

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
  \[ \delta(q, \sigma_1, \ldots, \sigma_k) = (p, p_1, D_1, \ldots, p_k, D_k). \]
  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 $p_1$ and move $D_1$ and so on.
- A configuration is defined as a $2k+1$-tuple $(q, w_1, u_1, \ldots, w_k, u_k)$.
- A k-string machine with input $x$ starts from the configuration $(s, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon)$.
- Relations $\rightarrow$, $\rightarrow^*$ are defined in analogy to ordinary machines.

---

**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:
  \[ \delta: K \times \Sigma^k \rightarrow (K \cup \{h, "yes", "no"\}) \times (\Sigma \times \{\rightarrow, =, \leftarrow\})^k \]
- This definition yields an ordinary Turing machine when $k = 1$. 

---

**Output is defined as for ordinary machines:**

If $(s, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon) \xrightarrow{M^*} ("yes", w_1, u_1, \ldots, w_k, u_k)$, then $M(x) = "yes"$.

If $(s, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon) \xrightarrow{M^*} ("no", w_1, u_1, \ldots, w_k, u_k)$, then $M(x) = "no"$.

If $(s, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon) \xrightarrow{M^*} (h, w_1, u_1, \ldots, w_k, u_k)$, then $M(x) = y$ where y is $w_k u_k$ with the leading $\triangleright$ and trailing $\varepsilon$s removed.

(Output read from the last (kth) string.)

**The time required** by $M$ on input $x$ is $t$ iff $(s, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon) \xrightarrow{(H, w_1, u_1, \ldots, w_k, u_k)}$ where $H \in \{h, "yes", "no"\}$.

If $M(x) = \bot$, then the time required is thought to be $\infty$. 

© 2007 TKK, Laboratory for Theoretical Computer Science
**Complexity classes**

- Performance is 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 \( \Sigma' = \Sigma \cup \{\sigma', <\} \).
- \( M' \) represents a configuration of \( M \) by concatenation
  \[ (q, w_1, u_1, \ldots, w_k, u_k) \mapsto (q, \sigma', w_1', u_1 <, w_2', u_2 <, \ldots, w_k', u_k <) \]
  where each \( w_i' \) is \( w_i \) with the leading \( > \) replaced by \( \sigma' \) and the last symbol \( \sigma \) by \( < \) to keep track of cursor positions.
- Initial configuration: \((s, >, \sigma', x <, \sigma' <, \ldots, \sigma' <, <)\)

**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 \( O(k^2f(n)) \) steps.
- Since \( M \) makes at most \( f(n) \) steps, \( M' \) makes \( O(f(n)^2) \) steps (\( k \) is fixed and independent of \( x \)).

\( \text{☞ Thesis: No conceivable “realistic” improvement on the Turing machine will increase the domain of the language such machines decide, or will affect their speed more than polynomially.} \)
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 \(|x| + 2 + 6[f(|x|)/m] \).
➤ The speedup is obtained if \( m = \lceil 6/\varepsilon \rceil \).
➤ Notice that a lot of new states have to be added: \(|K| \cdot m^k |\Sigma|^{3mk} \).

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 \( f'(n) \leq n^2 \)
(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!
➤ 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 consumption

Definition. Suppose that for a $k$-string Turing machine $M$ and an input $x$, $(x, \triangleright, x, \triangleright, \varepsilon, \ldots, \varepsilon) \xrightarrow{M^*} (H, w_1, u_1, \ldots, w_k, u_k)$ where $H \in \{"yes", "no", h\}$ is a halting state.

Then the space required by $M$ on input $x$ is $\sum_{i=1}^{k} |w_i| u_i$.

If $M$ is a Turing machine with input and output, then the space required by $M$ on input $x$ is $\sum_{i=2}^{k} |w_i| u_i$.

Let $f : \mathbb{N} \rightarrow \mathbb{N}$.

Turing machine $M$ operates within space bound $f(n)$ if for any input $x$, $M$ requires space at most $f(|x|)$.

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 $L$.

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

☞ 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 $L$. 

© 2007 TKK, Laboratory for Theoretical Computer Science