## Linear Speedup

- When using Turing machines, rate of growth of the time requirements is important but multiplicative and additive constants are not.
- In practice this also holds to some extent because of improving hardware.

**Theorem.** Let $L \in \mathbf{TIME}(f(n))$. Then for any $\varepsilon > 0$, $L \in \mathbf{TIME}(f'(n))$ where $f'(n) = \varepsilon f(n) + n + 2$.





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: adding word length
  Each symbol of $M'$ encodes several symbols of $M$ and each move of $M'$ several moves of $M$.
  The linear term comes from the initialization phase.
- Given $M$ and $\varepsilon$ we take some integer $m$ and use $m$ tuples of symbols of $M$ in $M'$.





(Proof—cont'd)

- $M'$ simulates $m$ steps of $M$ in at most a constant (6) amount of steps in a stage.
- In such a stage $M'$ reads the adjacent symbols ($m$-tuples) on both sides of the cursor (this takes 4 steps).
  The state of $M'$ "remembers" 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 $\varepsilon = \lceil 6/m \rceil$.
- Notice that a lot of new states have to be added: $|K| * m|\Sigma|^{3mk}$





## Linear Speedup—cont'd

- The theorem implies that 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 $f'(n) \approx \varepsilon n^2$ (*arbitrary linear speedup*).
- If $L$ is polynomially decidable, then $L \in \mathbf{TIME}(n^k)$ for some integer $k > 0$.

**Definition.** The set $\mathbf{P}$ of all languages decidable in polynomial time by Turing machines:

$$\mathbf{P} = \bigcup_{k>0} \mathbf{TIME}(n^k)$$

## Space bounds

- Space used by a computation: the sum of lengths of the final strings (strings cannot become shorter).
- There is an overcharge: sublinear space bounds impossible!
- Turing machines with read-only input and write-only output





## Space bounds—cont'd

**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 $\delta(q, \sigma_1, \dots, \sigma_k) = (p, \rho_1, D_1, \dots, \rho_k, D_k)$, then
(a) $\rho_1 = \sigma_1$
(b) $D_k \neq \leftarrow$
(c) if $\sigma_1 = \sqcup$, then $D_1 = \leftarrow$

**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 bounds—cont'd

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

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

If $M$ is a machine *with input and output*, then the space required by $M$ on input $x$ is $\Sigma_{i=2}^{k-1} |w_i u_i|$ .

Let $f : \mathbf{N} \mapsto \mathbf{N}$.
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* $\mathbf{SPACE}(f(n))$ is a set of *languages* $L$ such that $L$ is decided by a Turing machine with input and output operating within space bound $f(n)$.

**Definition.** $\mathbf{L} = \mathbf{SPACE}(\log(n))$.

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

## Random access machines

- Can Turing machines implement arbitrary algorithms?
- Conjecture (a strengthening of Church's thesis):
  Any reasonable attempt to model mathematically computer algorithms and their time performance ends up with a model of computation and associated time cost that is equivalent to Turing machines within a polynomial.
- Further evidence: simulation of random access machines.
  RAM is an idealization of a computer with arbitrarily large integers.





## Random Access Machine (RAM)

- Data structure: an array of registers, each capable of containing an arbitrarily large integer, possibly negative.
- A RAM program $\Pi = (\pi_1, \pi_1, \ldots, \pi_m)$ is a finite sequence of instructions (assembler language type).
- Register 0: accumulator
- Three modes of addressing: '$j$' / '$\uparrow j$' / '$= j$'
- Input is contained in a finite array of input registers $I = (i_1, \ldots, i_n)$.





## RAM—cont'd

| Instruction | Op | Semantics | Instr | Op | Semantics |
|---|---|---|---|---|---|
| READ | $j$ | $r_0 := i_j$ | JUMP | $j$ | $\kappa := j$ |
| READ | $\uparrow j$ | $r_0 := i_{r_j}$ | JPOS | $j$ | if $r_0 > 0,\ \kappa := j$ |
| STORE | $j$ | $r_j := r_0$ | JZERO | $j$ | if $r_0 = 0,\ \kappa := j$ |
| STORE | $\uparrow j$ | $r_{r_j} := r_0$ | JNEG | $j$ | if $r_0 < 0,\ \kappa := j$ |
| LOAD | $x$ | $r_0 := x$ | HALT | | $\kappa := 0$ |
| ADD | $x$ | $r_0 := r_0 + x$ | | | |
| SUB | $x$ | $r_0 := r_0 - x$ | where $x$ could be | | |
| HALF | | $r_0 := \lfloor \frac{r_0}{2} \rfloor$ | '$j$' / '$\uparrow j$' / '$= j$' | | |





## RAM—cont'd

- A configuration is a pair $C = (\kappa, R)$ where $\kappa$ is the instruction to be executed and $R = \{(j_1, r_{j_1}), (j_2, r_{j_2}), \ldots, (j_k, r_{j_k})\}$ is a finite set of register-value pairs. Initial configuration: $(1, \emptyset)$.
- For a RAM program $\Pi$ and an input $I = (i_1, \ldots, i_n)$, $(\kappa, R) \xrightarrow{\Pi,I} (\kappa', R')$ (yields in one step):
  $\kappa'$ is the new value of $\kappa$ after the execution of the $\kappa$th instruction of $\Pi$ and $R'$ is $R$ with possibly some pair $(j, x)$ deleted and $(j', x')$ added according to the $\kappa$th instruction of $\Pi$.
  This induces $\xrightarrow{\Pi,I^k}$ and $\xrightarrow{\Pi,I^*}$ .
- Let $D$ be a set of finite sequences of integers and $\phi : D \mapsto \mathbf{N}$.
  A RAM $\Pi$ *computes* $\phi$ iff for any $I \in D$, $(1, \emptyset) \xrightarrow{\Pi,I^*} (0, R)$ with $(0, \phi(I)) \in R$.

| **Example.** | Program: | Configurations: |
|---|---|---|
| Input: | 1. READ 2 | (1, {}) |
| $I = (6, 10)$ | 2. STORE 2 | (2, {(0,10)}) |
| | 3. READ 1 | (3, {(0,10), (2,10)}) |
| $\phi(I) = 4$ | 4. STORE 1 | (4, {(0,6), (2,10)}) |
| | 5. SUB 2 | (5, {(0,6), (2,10), (1,6)}) |
| | 6. JNEG 8 | (6, {(0,-4), (2,10), (1,6)}) |
| | 7. HALT | (8, {(0,-4), (2,10), (1,6)}) |
| | 8. LOAD 2 | (9, {(0,10), (2,10), (1,6)}) |
| | 9. SUB 1 | (10, {(0,4), (2,10), (1,6)}) |
| | 10.HALT | (0, {(0,4), (2,10), (1,6)}) |





## RAM—cont'd

- Time requirements: each RAM operation as a single step.
  For example, addition of large integers in one step.
  However, multiplication is not included in the instruction set.
- The size of the input through logarithms:
  For an integer $i$, $\mathrm{b}(i)$ is its binary representation with no redundant leading 0s and with a minus sign in front if negative.
  The length of integer $i$, $\mathrm{l}(i) = |\mathrm{b}(i)|$.
  For a sequence of integers $I = (i_1, \ldots, i_n)$, $\mathrm{l}(I) = \Sigma_{j=1}^{n} \mathrm{l}(i_j)$.





## RAM—cont'd

- Suppose a RAM program $\Pi$ computes a function $\phi$ from $D$ to integers and let $f$ be a function from nonneg. integers $\mathbf{N}^+$ to $\mathbf{N}^+$.
  $\Pi$ computes $\phi$ *in time $f(n)$* iff for any $I \in D$, $(1, \emptyset) \xrightarrow{\Pi, I^k} (0, R)$ where $k \leq f(\mathrm{l}(I))$.
- RAM programs are powerful.
  For example, multiplication of arbitrarily large integers in linear number of steps (i.e., the number of steps is propositional to the logarithm of the input integers).





## RAM—cont'd

- Simulation of a Turing machine with linear loss of efficiency:
  Let $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ be the alphabet of the Turing machine.
  $D_\Sigma = \{(i_1, \ldots, i_n, 0) \mid n \geq 0, 1 \leq i_j \leq k, j = 1, \ldots, n\}$
  If $L \subset (\Sigma - \{\sqcup\})^*$, define $\phi_L : D_\Sigma \mapsto \{0, 1\}$ with $\phi_L(i_1, \ldots, i_n, 0) = 1$ iff $\sigma_{i_1} \cdots \sigma_{i_n} \in L$ and 0 otherwise.
  (Deciding $L$ is equivalent to computing $\phi_L$.)

**Theorem.** Let $L \in \mathbf{TIME}(f(n))$. Then there is a RAM program which computes the function $\phi_L$ in time $O(f(n))$.

Proof sketch. Construct a subroutine in RAM to simulate each state of the Turing machine $M$ deciding $L$.

## TM simulating RAM

- Any RAM can be simulated by a Turing machine with only a polynomial loss of efficiency.
- Input-output conventions:
  Let $\phi$ be a function from $D$ to the integers where $D$ is a set of finite sequences of integers.
  The binary representation of $I = (i_1, \dots, i_n)$, denoted by $\mathrm{b}(I)$, is the string $\mathrm{b}(i_1); \dots; \mathrm{b}(i_n)$.
- A Turing machine $M$ computes $\phi$ if, for any $I \in D$, $M(\mathrm{b}(I)) = \mathrm{b}(\phi(I))$.





## TM simulating RAM—cont'd

**Theorem.** If a RAM program computes a function $\phi$ in time $f(n)$, then there is a 7-string Turing machine $M$ which computes $\phi$ in time $O(f(n)^3)$.

Proof sketch.

- String 1: input
  String 2: representation of register contents $\dots; \mathrm{b}(i) : \mathrm{b}(r_i); \dots \triangleleft$
  (update: erase old value by $\sqcup$s and add new value to the right)
  String 3: program counter
  String 4: register address current sought
  Strings 5–7: operations





(Proof—cont'd)

- Each instruction of the RAM program is implemented by a group of states of $M$.
- Simulating an instruction of $\Pi$ by $M$ takes $O(f(n)l)$ steps where $l$ is the size of the largest integer in the registers (as there are $O(f(n))$ pairs on string 2).
- Simulating $\Pi$ by $M$ takes $O(f(n)^2 l)$ steps.
  It remains to show that $l = O(f(n))$.

**Claim:** After the $t$th step of a RAM program $\Pi$ computation on input $I$, the contents of any register have length at most $t + \mathrm{l}(I) + \mathrm{l}(B)$ where $B$ is the largest integer referred to in $\Pi$.





Proof of the claim by induction on $t$:

- The claim holds when $t = 0$.
  Suppose it is true up to the $t$th step.
- Case analysis over instruction types of the $t$th instruction:
  Most of the instruction do no create new values (jumps, HALT, LOAD, STORE, READ). For these the claim continues to hold after the execution of the instruction.
  Consider arithmetic, say ADD, involving two integers $i$ and $j$.
  The length of the result is one plus the length of the longest operand which is by induction hypothesis at most $t - 1 + \mathrm{l}(I) + \mathrm{l}(B)$. Hence the result has length at most $t + \mathrm{l}(I) + \mathrm{l}(B)$.

## Nondeterministic machines

- Nondeterministic machines are an unrealistic model of computation.
- Can be simulated by deterministic Turing machines with an exponential loss of efficiency.
- An open question: is polynomial simulation possible? ($\mathbf{P} = \mathbf{NP}$?)





## Nondeterministic machines—cont'd

**Definition.** A nondeterministic Turing machine is a quadruple $M = (K, \Sigma, \Delta, s)$ like the ordinary Turing machine except that $\Delta$ is a *relation* (instead of a function):
$\Delta \subset (K \times \Sigma) \times [(K \cup \{\text{h}, \text{“yes”}, \text{“no”}\}) \times \Sigma \times \{\rightarrow, \leftarrow, -\}]$

- A configuration is defined as before but “yields” is no longer a function but a relation:
  $(q, w, u) \xrightarrow{N} (q', w', u')$ iff there is a rule in $\Delta$ to make this a legal transition.
- Power of nondeterminism: weak input-output behavior





## Nondeterministic machines—cont'd

**Definition.** A nondeterministic Turing machine $N$ decides a language $L$ iff for any $x \in \Sigma^*$, the following holds:

$x \in L$ iff $(s, \triangleright, x) \xrightarrow{N^*} (\text{“yes”}, w, u)$ for some strings $w$ and $u$.

Note: (i) An input is accepted if there is some sequence of nondeterministic choices that results in “yes”.
(ii) The input is rejected only if no sequence of nondeterministic choices can lead to acceptance.





## Nondeterministic machines—time complexity

**Definition.** A nondeterministic Turing machine $N$ decides a language $L$ in time $f(n)$ iff $N$ decides $L$ and for any $x \in \Sigma^*$,

if $(s, \triangleright, x) \xrightarrow{N^k} (q, w, u)$, then $k \leq f(|x|)$.

(Note: all computation paths should have length at most $f(|x|)$.)

**Definition.** A *complexity class* $\mathbf{NTIME}(f(n))$ is a set of *languages* $L$ such that $L$ is decided by a nondeterministic Turing machine in time $f(n)$.

## Nondeterministic machines—time complexity

**Definition.** The set **NP** of all languages decidable in polynomial time by a nondeterministic Turing machine:

$$\mathbf{NP} = \bigcup_{k>0} \mathbf{NTIME}(n^k)$$

Note: $\mathbf{P} \subseteq \mathbf{NP}$.

**Theorem.** Let language $L$ be decided by a nondeterministic Turing machine $N$ in time $f(n)$. Then it is decided by a 3-string deterministic Turing machine $M$ in time $O(c^{f(n)})$, where $c > 1$ is some constant depending on $N$.

$$\mathbf{NTIME}(f(n)) \subseteq \bigcup_{c>1} \mathbf{TIME}(c^{f(n)})$$





Proof sketch:

- Let $N = (K, \Sigma, \Delta, s)$ be a NTM.
  Let $d$ be the degree on nondeterminism (maximal number of possible moves for any state-symbol pair in $\Delta$).
- Any computation of $N$ is a sequence of nondeterministic choices.
  A sequence of $t$ choices is a sequence of $t$ integers from $0, 1, \ldots, d-1$.
- The simulating machine $M$ considers all such sequences of choices in order of increasing length and simulates $N$ on each.
- With sequence $(c_1, c_2, \ldots, c_t)$ $M$ simulates the actions that $N$ would have taken had $N$ taken choice $c_i$ at step $i$ for its first $t$ steps.





(Proof—cont'd)

- If a sequence leads $N$ to halting with "yes", then $M$ does, too. Otherwise it considers the next sequence.
- When to reject (bound $f(n)$ is not known)?
  $M$ simulates all sequences of choices of length $t$ and finds no continuing computation.
- Time bound: $O(c^{f(n)})$
  Number of sequences: $\Sigma_{t=1}^{f(n)} d^t = O(d^{f(n)+1})$.
  Cost of a sequence: $O(2^{f(n)})$.





## Nondeterministic machines—space complexity

- For space, a nondeterministic Turing machine with input and output is needed.
- Given a $k$-string nondeterministic Turing machine $N$ with input and output, we say that $N$ decides language $L$ within space $f(n)$ if $N$ decides $L$ and for any $x \in (\Sigma - \{\sqcup\})^*$, if $(s, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon,) \xrightarrow{N^*} (q, w_1, u_1, \ldots, w_k, u_k)$, then $\Sigma_{i=2}^{k-1} |w_i u_i| \leq f(|x|)$.
- REACHABILITY can be solved nondeterministically within space $O(\log n)$.

## Universal Turing machine

- A Turing machine (RAM) has a fixed program, i.e., solves a single problem.
- A *universal Turing machine* $U$ takes as input a description of another Turing machine, $M$, and an input, say $x$.
  $U$ simulates $M$ on $x$, i.e., $U(M;x) = M(x)$.
- Coding a Turing machine $M = (K, \Sigma, \delta, s)$ using integers:
  $\Sigma = \{1, 2, \ldots, |\Sigma|\}$
  $K = \{|\Sigma|+1, |\Sigma|+2, \ldots, |\Sigma|+|K|\}$
  $s = |\Sigma|+1$
  $|\Sigma|+|K|+1, |\Sigma|+|K|+2, \ldots, |\Sigma|+|K|+6$ encode
  $\leftarrow, \rightarrow, -, \mathrm{h}$, "yes", "no", respectively.





- A machine $M = (K, \Sigma, \delta, s)$ is encoded as $\mathrm{b}(|\Sigma|);\mathrm{b}(|K|);\mathrm{e}(\delta)$ where all number, say $|\Sigma|$, are represented as $\mathrm{b}(|\Sigma|)$ in binary with exactly $\lceil \log(|\Sigma|+|K|+6) \rceil$ bits and
  $\mathrm{e}(\delta)$ is a sequence of pairs $((q,\sigma),(p,\rho,D))$ describing the transition function $\delta$.
  The symbol $M$ is used to denote this description of machine $M$.
- $U$ simulates $M$ using two strings:
  The first string $S_1$: the description of $M$.
  The second string $S_2$: the current configuration $(q,w,u)$ of $M$ (encoded as $w,q,u$). Initially: $\triangleright, s, x$.
  Simulation of a step:
  (i) Scan $S_2$ to find an integer corresponding to a state.
  (ii) Search $S_1$ for a rule of $\delta$ matching the current state.
  (iii) Implement the rule. (When $M$ halts, so does $U$.)





## Undecidability—halting problem

- There are more languages than machines for deciding them. Hence, undecidable problems must exist.
- HALTING: Given the description of a Turing machine $M$ and its input $x$, will $M$ halt on $x$?
  The corresponding language $H = \{M;x \mid M(x) \neq \nearrow\}$
- HALTING turns out to be undecidable, i.e., there is no Turing machine deciding $H$.





## Halting problem—cont'd

- $H$ is recursively enumerable.
- $H$ is a complete for r.e. languages:
  any r.e. language $L$ can be reduced to it.
  Proof: Let $M_L$ be the machine accepting $L$.
  Then $x \in L$ iff $M_L$ halts given $x$ as input iff $M_L;x \in H$.

## Halting problem—cont'd

**Theorem.** $H$ is not recursive.

**Proof sketch.**

- Assume that $H$ is recursive, i.e., $M_H$ decides $H$.
- Consider the machine $D(M)$:
  **if** $M_H(M;M) =$ "yes" **then** $\nearrow$ **else** "yes".
- If $D(D) = \nearrow$, then $M_H(D;D) =$ "yes", i.e., $D(D) \neq \nearrow$, a contradiction.
  Hence, $D(D) \neq \nearrow$. Then $M_H(D,D) \neq$ "yes". But as $M_H$ decides $H$, $M_H(D,D) =$ "no" and, hence, $D;D \notin H$, i.e. $D(D) = \nearrow$, a contradiction again. Thus, $H$ is not recursive.





## More undecidability

- HALTING spawns a range of other undecidable problems using the *reduction technique*.
- To show a problem $A$ undecidable, establish that if there is an algorithm for $A$, then there is an algorithm for HALTING.
- This can be shown by devising a *reduction from* HALTING to $A$, i.e., a transformation of the input $M;x$ of HALTING to the input $t(M;x)$ of $A$ such that
  $M;x \in H$ iff $t(M;x) \in A$.





## Undecidability—cont'd

The following languages are not recursive:

(a) $\{M \mid M \text{ halts on all inputs}\}$

(b) $\{M;x \mid \text{there is } y \text{ such that } M(x) = y\}$

(c) $\{M;x \mid \text{the computation } M \text{ on input } x \text{ uses all states of } M\}$

(d) $\{M;x;y \mid M(x) = y\}$

Proof sketch for (a):
Reduction of HALTING to this problem $A$: Given $M;x$, we construct a machine $M'(y)$: **if** $x = y$ **then** $M(x)$ **else halt**.

$M;x \in H$ iff $M$ halts on $x$ iff $M'$ halts on all inputs iff $M' \in A$





## Undecidability—cont'd

**Proposition.** If $L$ is recursive, then so is $\overline{L}$ (the complement of $L$).

**Proposition.** $L$ is recursive iff both $L$ and $\overline{L}$ are recursively enumerable.

Proof sketch. ($\Rightarrow$) By previous proposition.
($\Leftarrow$) Simulate $M_L$ and $M_{\overline{L}}$ on input $x$ in an interleaved fashion:
If $M_L$ accepts, return "yes" and if $M_{\overline{L}}$ accepts, return "no".

☞ *$\overline{H}$ is not recursively enumerable.*

## Undecidability—cont'd

**Proposition.** $L$ is *recursively enumerable* iff there is a machine $M$ such that $L = E(M) = \{x \mid (s, \triangleright, \varepsilon) \overset{M}{\rightarrow}{}^{*} (q, y \sqcup x \sqcup, \varepsilon)\}$

Any non-trivial property of Turing machines is undecidable:

**Theorem.** **(Rice's Theorem)** Let $C$ be a proper non-empty subset of r.e. languages. Then the following problem is undecidable: given a Turing machine $M$, is $L(M) \in C$?

($L(M)$ is the language accepted by a Turing machine $M$.)