MORE ABOUT TURING MACHINES

- Random access machines
- Nondeterministic machines
- Universal Turing machine
- Halting problem
- Undecidability

(C. Papadimitriou: *Computational complexity*, Chapters 2.6–3.3)

1. 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: Turing machines can simulate *random access machines* (RAMs) which idealize a computer capable of handling arbitrarily large integers.

Basic definitions of RAMs

- Data structure: an array of registers, each capable of containing an arbitrarily large integer, possibly negative.
- A RAM program $\Pi = (\pi_1, \pi_2, \ldots, \pi_m)$ is a finite sequence of *instructions* (of assembler language type).
- Register 0 serves as an accumulator
- Three modes of addressing: 'j' / '$\uparrow j$' / '$= j$'
- Input is contained in a finite array of input registers $I = (i_1, \ldots, i_n)$.

Instruction set

Instruction	Op	Semantics	Instruction	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	$r_0 := r_0 + x'$	where x (resp. x') is one of		
SUB	x	$r_0 := r_0 - x'$	'j' / '$\uparrow j$' / '$= j$'		
HALF		$r_0 := \lfloor \frac{r_0}{2} \rfloor$	(resp. r_j / r_{r_j} / j)		

Computations through configurations

- A configuration is a pair $C = (\kappa, R)$ where κ is the number of 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.
  The initial configuration: $(1, \emptyset)$.
- For a RAM program Π and an input $I = (i_1, \ldots, i_n)$, the relation $(\kappa, R) \overset{\Pi,I}{\rightarrow} (\kappa', R')$ (yields in one step) is defined as follows:
  – κ' is the new value of κ after executing the κth instruction of Π,
  – R' is R with possibly some pair (j, x) deleted and (j', x') added according to the κth instruction of Π.
- The relation $\overset{\Pi,I}{\rightarrow}$ induces $\overset{\Pi,I^k}{\rightarrow}$ and $\overset{\Pi,I^*}{\rightarrow}$ as previously.

Definition. Let D be a set of finite sequences of integers. A RAM Π *computes* $\phi : D \rightarrow \mathbf{Z}$ iff $\forall I \in D$, $(1, \emptyset) \overset{\Pi,I^*}{\rightarrow} (0, R)$ so that $(0, \phi(I)) \in R$.

Example. A RAM programming computing $\phi(x,y) = |x - y|$

Input:	Program:	Configurations:
$I = (6, 10)$	1. READ 2	(1, {})
	2. STORE 2	(2, {(0,10)})
$\phi(I) = 4$	3. READ 1	(3, {(0,10), (2,10)})
	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)})

Counting time and space

- The execution of each RAM instruction counts as one time step.
  – Addition of large integers takes place in one step.
  – Multiplication is not included in the instruction set.
- The size of the input is computed in terms of 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 $I = (i_1, \ldots, i_n)$, $\mathrm{l}(I) = \Sigma_{j=1}^{n} \mathrm{l}(i_j)$.

Time bounds for RAMs

Definition. Suppose that a RAM program Π computes a function $\phi : D \rightarrow \mathbf{Z}$ and let $f : \mathbf{N}^+ \rightarrow \mathbf{N}^+$.

The program Π computes ϕ *in time $f(n)$* iff

$$\text{for any } I \in D,\ (1, \emptyset) \overset{\Pi,I^k}{\rightarrow} (0, R) \text{ so that } k \leq f(\mathrm{l}(I)).$$

Example. The multiplication of arbitrarily large integers is accomplished by a RAM in linear number of steps (i.e., the number of steps is propositional to the logarithm of the input integers).

☞ RAM programs are powerful.

Example. A RAM for solving REACHABILITY can be found in the textbook.

Simulating TMs with RAMs

- The simulation of a Turing machine having an alphabet $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ is possible with a linear loss of efficiency.
- The domain of inputs for the simulating RAM is $D_\Sigma = \{(i_1, \ldots, i_n, 0) \mid n \geq 0,\ 1 \leq i_j \leq k,\ j = 1, \ldots, n\}$.
- For a language $L \subset (\Sigma - \{\sqcup\})^*$, define $\phi_L : D_\Sigma \mapsto \{0,1\}$ so that

$$\phi_L(i_1, \ldots, i_n, 0) = 1 \text{ iff } \sigma_{i_1} \cdots \sigma_{i_n} \in L.$$

☞ Deciding L is equivalent to computing ϕ_L.

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

Proof sketch. Construct a subroutine (a RAM program) which simulates each state transition of the Turing machine M deciding L.

Simulating RAMs with TMs

- Any RAM can be simulated by a Turing machine with only a polynomial loss of efficiency.
- The binary representation of a sequence $I = (i_1, \ldots, i_n)$ of integers, denoted by $\mathrm{b}(I)$, is the string $\mathrm{b}(i_1); \ldots; \mathrm{b}(i_n)$.

Definition. Let D be a set of finite sequences of integers and $\phi : D \to \mathbf{Z}$. A Turing machine M computes ϕ iff for any $I \in D$,

$$M(\mathrm{b}(I)) = \mathrm{b}(\phi(I)).$$

Simulating RAMs with TMs

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

Proof sketch.

The strings of the machine serve the following purposes:

1. Input
2. Representation of register contents $\ldots; \mathrm{b}(i) : \mathrm{b}(r_i); \ldots \triangleleft$
   (update: erase old value by $\sqcup$s and add new value to the right)
3. Program counter
4. Register address currently sought

5.–7. Extra space reserved for the execution of instructions

Proof skecth—cont'd.

- Each instruction of the RAM program is implemented by a group of states of M.
- Simulating an instruction of Π on 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 Π on M takes $O(f(n)^2 l)$ steps.
- It remains to establish that $l = O(f(n))$.

Claim: After the tth step of a RAM program Π 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 an instruction of Π.

Inductive proof of the claim

- Base case: the claim is true when $t = 0$.
- Induction hypothesis: the claim is true after the (t-1)th step.
- Case analysis over instruction types of the tth 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 $t+\mathrm{l}(I)+\mathrm{l}(B)$.

2. Nondeterministic Machines

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

Transition relation

Definition. A nondeterministic Turing machine (NTM) is a quadruple $N = (K, \Sigma, \Delta, s)$ like the ordinary Turing machine except that Δ is a *transition relation* (rather than a transition function):

$$\Delta \subset (K \times \Sigma) \times [(K \cup \{\text{h}, \text{“yes”}, \text{“no”}\}) \times \Sigma \times \{\rightarrow, \leftarrow, -\}]$$

- Configurations are defined as before but “yields” is a relation (rather than a function) for a NTM N: $(q, w, u) \xrightarrow{N} (q', w', u')$ iff there is a tuple in Δ that makes this a legal transition.
- The power of nondeterminism boils down to the weak input-output behavior demanded of NTMs.

NMTs deciding languages

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

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

Notes:

(i) An input is accepted if there is some sequence of nondeterministic choices that results in the accepting state “yes”.

(ii) The input is rejected only if no sequence of nondeterministic choices can lead to acceptance.

Time complexity classes

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

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

☞ All computation paths should have length at most $f(|x|)$.

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

Time complexity classes—cont'd

Definition. The set $\mathbf{NP}$ of all languages decidable in polynomial time by a nondeterministic Turing machine:

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

☞ $\mathbf{P} \subseteq \mathbf{NP}$ as TMs are also NTMs.

Theorem. Suppose that a language L is decided by a NTM N in time $f(n)$. Then it is decided by a 3-string deterministic TM M in time $\mathrm{O}(c^{f(n)})$, where $c > 1$ is some constant depending on N. Thus

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

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 Δ) for N.
- 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 if N had taken choice c_i at step i for its first t steps.

Proof sketch—cont'd

- If a sequence leads N to halting with "yes", then M does, too, and considers the next sequence otherwise.
- The machine M rejects its input whenever it has simulated all sequences of choices of length t and each of them ends with halting in state "no" or h.

  Note that the bound $f(n)$ is not available to M!
- The time bound $\mathrm{O}(c^{f(n)})$ is then established as the product of

  – the number of sequences $\Sigma_{t=1}^{f(n)} d^t = \mathrm{O}(d^{f(n)+1})$ and

  – the cost of each sequence which is $\mathrm{O}(2^{f(n)})$.

Space complexity

- For space considerations, a nondeterministic Turing machine with input and output is needed.
- Given a k-string NTM 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 $\Sigma_{i=2}^{k-1}|w_i u_i| \leq f(|x|)$.

Example. REACHABILITY is nondeterministically solvable within space $\mathrm{O}(\log n)$.

3. Universal Turing Machine

- A TM has a fixed program which solves a single problem.
- A *universal Turing machine* U takes as input a description of another Turing machine M and an input x for M, and then simulates M on x so that $U(M;x) = M(x)$.
- Encoding 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.

Encoding TMs using integers

- An entire TM $M = (K,\Sigma,\delta,s)$ is encoded as $\mathrm{b}(|\Sigma|);\mathrm{b}(|K|);\mathrm{e}(\delta)$ where all integers i are represented as $\mathrm{b}(i)$ 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 δ.

  (The symbol M is also used to denote this description of M).
- Then U simulates M using a string S_1 for the description of M and a string S_2 for the current configuration (q,w,u) of M.

  Simulation of a step of M is performed as follows:
  (i) Scan S_2 to find an integer corresponding to a state.
  (ii) Search S_1 for a rule of δ matching the current state.
  (iii) Implement the rule. (When M halts, so does U.)

4. Halting Problem

- There are more languages than TMs for deciding them.

  ☞ 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 is defined as

  $$H = \{M;x \mid M(x) \neq \nearrow\}.$$
- HALTING turns out to be an undecidable language, i.e., there is no Turing machine deciding H.

Properties of HALTING

- HALTING is recursively enumerable (r.e. for short).

  Proof: A slight variant U' of the universal Turing machine U *accepts* H: all halting states of U are forced to be "yes" states.

  1. If $M;x \in H$, then $M(x) \neq \nearrow$, $U(M,x) \neq \nearrow$, $U'(M,x) =$ "yes".
  2. If $M;x \notin H$, then $M(x) = U(M,x) = U'(M,x) = \nearrow$.

- HALTING is *complete* for r.e. languages, i.e. any r.e. language L can be reduced to it.

  Proof: Let M_L be the machine *accepting* L.

  Then $x \in L$ iff $M_L(x) =$ "yes" iff $M_L(x) \neq \nearrow$ iff $M_L;x \in H$.

- HALTING is not recursive.

Proof sketch

- Assume that H is recursive, i.e., some M_H *decides* H.
- Consider the following TM D operating on an input M:

  if $M_H(M;M) =$ "yes" **then** $\nearrow$ **else** "yes".

- There is no satisfactory answer to D(D):

  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.

☞ H is not recursive.

5. Undecidability

- HALTING spawns a range of other undecidable problems using a *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

$$M;x \in H \text{ iff } t(M;x) \in A.$$

  (It is assumed above that $t(M;x) = T(M;x)$ for some TM T.)

Further undecidable languages

The following languages are not recursive:

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

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

(c) $\{M;x \mid$ the computation M on input x 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**.

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

Properties of recursive languages

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

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

Proof sketch.

($\Rightarrow$) By previous proposition and the fact that every recursive language is also recursively enumerable.

($\Leftarrow$) 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".

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

Recursively enumerable languages

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

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 $L(M) \in C$?

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

Learning Objectives

➤ You should be able to justify why Turing machines make a powerful model of algorithms/computation.

➤ Basic understanding of differences between deterministic and nondeterministic Turing machines.

➤ The definitions and background of complexity class **NP** and the problem whether **P** = **NP** or not.

➤ The definitions of recursive and recursively enumerable languages (including examples of such languages).