Computational Complexity

Pascal Hitzler
http://www.pascal-hitzler.de

CS 7220 Lecture, Spring 2016
Wright State University, Dayton, OH, U.S.A.

Version: 2016-04-05 (tentative, incomplete manuscript)

Contents

[**Slideset 1: Motivation**]

1 Big-Oh-Notation

[4, Chapter 14.2]

$\mathbb{N} = \{0, 1, 2, \dots\}$
$\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$
$\mathbb{R}$: the real numbers

1.1 Definition
$f, g : \mathbb{N} \to \mathbb{N}$
f is *of order* g, if there is a constant $c > 0$ and $n_0 \in \mathbb{N}$ s.t. $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.
$O(g) := \{f \mid f \text{ is of order } g\}$ ("big oh of g").
If $f \in O(g)$ we can say that g *provides an asymptotic upper bound on* f.
If $f \in O(g)$ and $g \in O(g)$, then they *have the same rate of growth*, and g *is an asymptotically tight bound on* f (and vice versa).

1.2 Remark
Common abuse of notation:
$f = O(g)$ instead of $f \in O(g)$.
$f(n) = n^2 + O(n)$ instead of "$f(n) = n^2 + g(n)$ for some $g \in O(n)$".

1.3 Example
$f(n) = n^2$; $g(n) = n^3$
$f \in O(g)$. [For $c = 1$ and $n > 1$, $n^2 \leq c \cdot n^3$.]
$g \notin O(f)$.
[Assume $n^3 \in O(n^2)$. Then ex. c, n_0 s.t. $n^3 \leq c \cdot n^2$ for all $n \geq n_0$. Choose $n_1 = 1 + \max\{c, n_0\}$. Then $n_1^3 = n_1 \cdot n_1^2 > c \cdot n_1^2$ and $n_1 > n_0$. $\lightning$]

Exercise 1 Show that $2^n \in O(n!)$.

Exercise 2 Show that $n! \notin O(2^n)$.

Exercise 3 Show: If $f \in O(g)$ and $g \in O(h)$, then $f \in O(h)$.

1.4 Example
$f(n) = n^2 + 2n + 5$; $g(n) = n^2$
$g \in O(f)$ [For $c = 1$ and $n > 0$, $n^2 \leq c \cdot (n^2 + 2n + 5)$.]
$f \in O(g)$
[For $n > 1$ we have $f(n) = n^2 + 2n + 5 \leq n^2 + 2n^2 + 5n^2 = 8n^2 = 8 \cdot g(n)$. Hence, for $c = 8$ and $n > 1$, $f(n) \leq c \cdot g(n)$.]

1.5 Definition
$\Theta(g) := \{f \mid f \in O(g) \text{ and } g \in O(f)\}$

1.6 Example
For f, g from Example 1.4, $f \in \Theta(g)$.

1.7 Remark
A *polynomial* (with integer coefficients) *of degree* $r \in \mathbb{N}$ is a function of the form

$$f : \mathbb{N} \to \mathbb{Z} : n \mapsto c_r \cdot n^r + c_{r-1} \cdot n^{r-1} + \cdots + c_1 \cdot n + c_0,$$

with $0 < r \in \mathbb{N}$, *coefficients* $c_i \in \mathbb{Z}$ $(i = 1, \ldots, r)$, $c_r \neq 0$.
For $f, g : \mathbb{N} \to \mathbb{Z}$, we say $f \in O(g)$ if $|f| \in O(|g|)$, where $|f| : \mathbb{N} \to \mathbb{N} : n \mapsto |f(n)|$.

1.8 Example
$f(n) = n^2 + 2n + 5$; $g(n) = -n^2$
$g \in O(f)$
[We have $|g| : n \mapsto n^2$ and $|f| \equiv f$. Thus, from Example 1.4 we know that $|g| \in O(|f|)$.]
$f \in O(g)$ [As before, from Example 1.4.]

1.9 Remark
For $f, g : \mathbb{N} \to \mathbb{R}$, we say $f \in O(g)$ if $\lfloor f \rfloor \in O(\lfloor g \rfloor)$.

1.10 Remark
$\log_a(n) \in O(\log_b(n))$ for all $1 < a, b \in \mathbb{N}$. [$\log_a(n) = \log_a(b) \cdot \log_b(n)$ for all $n \in \mathbb{N}$.]

1.11 Theorem
The following hold.

1. If $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = 0$, then $f \in O(g)$ and $g \notin O(f)$.
2. If $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = c$ with $0 < c < \infty$, then $f \in \Theta(g)$ and $g \in \Theta(f)$.
3. If $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, then $f \notin O(g)$ and $g \in O(f)$.

Proof: We show part 1.
Assume $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = 0$, i.e., for each $\varepsilon > 0$ there exists $n_\varepsilon \in \mathbb{N}$ such that for all $n \geq n_\varepsilon$ we have $\frac{f(n)}{g(n)} < \varepsilon$, and hence $f(n) < \varepsilon g(n)$. Now select $c = \varepsilon = 1$ and $n_0 = n_\varepsilon$. Then $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$, which shows $f \in O(g)$.
Now if we also assume $g \in O(f)$, then there must exist $d > 0$ and $m_0 \in \mathbb{N}$ s.t. $g(n) \leq d \cdot f(n)$ for all $n \geq m_0$, i.e., $\frac{1}{d} \leq \frac{f(n)}{g(n)}$ for all $n \geq m_0$. But then $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} \geq \frac{1}{d} > 0$ ↯. ■

Exercise 4 Show Theorem 1.11 part 2.

Exercise 5 Show Theorem 1.11 part 3. [Hint: Use part 1.]

1.12 Remark
The l'Hospital's Rule often comes in handy:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)}$$

1.13 Example
$n \log_a(n) \in O(n^2)$ and $n^2 \notin O(n \log_a(n))$

$$\left[\lim_{n\to\infty} \frac{n \log_a(n)}{n^2} = \lim_{n\to\infty} \frac{\log_a(n) + n(\log_a(e)/n)}{2n} = \lim_{n\to\infty} \frac{\log_a(n)}{2n} + \lim_{n\to\infty} \frac{\log_a(e)}{2n} = 0 + 0 = 0\right]$$

1.14 Theorem
Let f be a polynomial of degree r. Then

(1) $f \in \Theta(n^r)$

(2) $f \in O(n^k)$ for all $k > r$

(3) $f \notin O(n^k)$ for all $k < r$

Exercise 6 Prove Theorem 1.14 (1).

1.15 Theorem
The following hold.

(1) $\log_a(n) \in O(n)$ and $n \notin O(\log_a(n))$

(2) $n^r \in O(b^n)$ and $b^n \notin O(n^r)$

(3) $b^n \in O(n!)$ and $n! \notin O(b^n)$

Exercise 7 Prove Theorem 1.15 (1).

1.16 Remark
The Big Oh Hierarchy.

$O(1)$	constant	"sublinear"	"subpolynomial"
$O(\log_a(n))$	logarithmic	"sublinear"	"subpolynomial"
$O(n)$	linear		"subpolynomial"
$O(n \log_a(n))$	$n \log n$		"subpolynomial"
$O(n^2)$	quadratic		"polynomial"
$O(n^3)$	cubic		"polynomial"
$O(n^r)$	polynomial ($r \geq 0$)		
$O(b^n)$	exponential ($b > 1$)		"exponential"
$O(n!)$	factorial		"exponential"

2 Turing Machines and Time Complexity

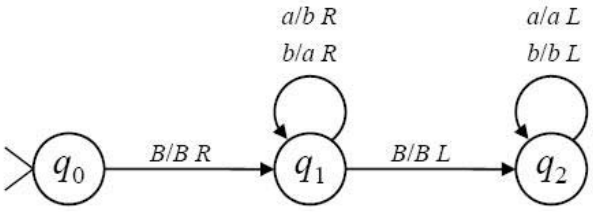
[4, Chapter 14.3 and recap Chapters 8.1, 8.2]

2.1 Definition
A standard (*single tape, deterministic*) *Turing machine* (TM) is a quintuple $M = (Q, \Sigma, \Gamma, \delta, q_0)$ with
Q a finite set of *states*
Γ a finite set called *tape alphabet* containing a *blank* B
$\Sigma \subseteq \Gamma \setminus \{B\}$ the *input alphabet*
$\delta : Q \times \Gamma \overset{\text{partial}}{\to} Q \times \Gamma \times \{L, R\}$, the *transition function*
$q_0 \in Q$ the *start state*

Recall:

- Tape has a left boundary and is infinite to the right.
- Tape positions numbered starting with 0.
- Each tape position contains an element from Γ.
- Machine starts in state q_0 and at position 0.
- Input is written on tape beginning at 1.
- Rest of tape is blank.
- A transition
  1. changes state,
  2. writes new symbol at tape position,
  3. moves head left or right.
- Computation halts if no transition is defined.
- Computation terminates abnormally if it moves left of position 0.
- TMs can be represented by state diagrams.

2.2 Example
Swap all a's to b's and all b's to a's in a string of a's and b's.

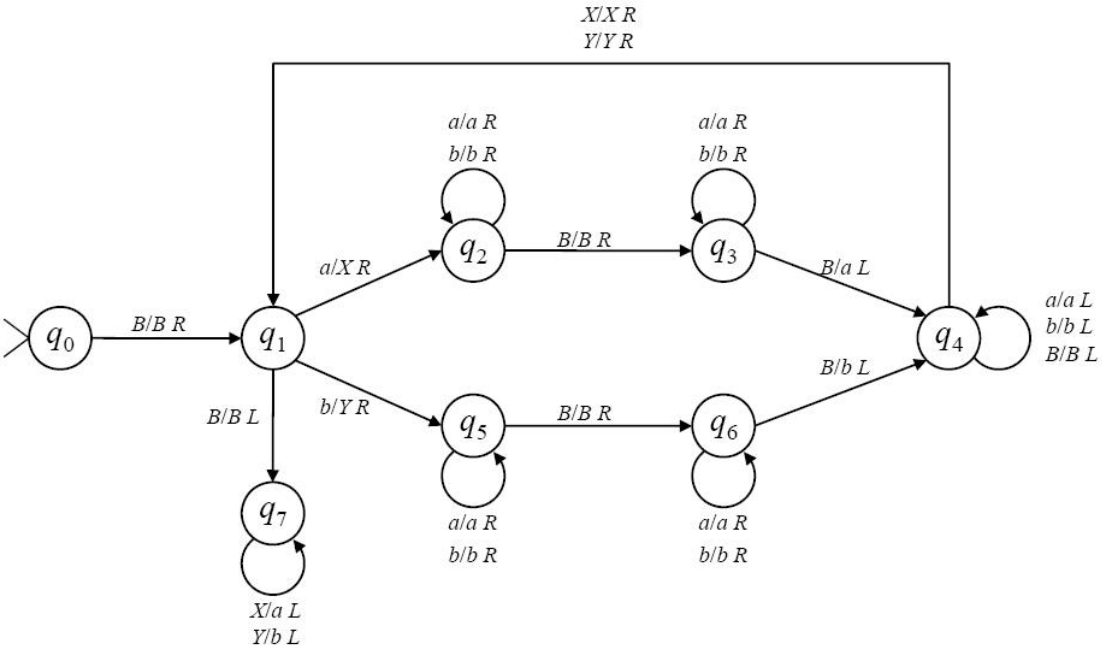


Computation example:

$$\begin{array}{lll} \vdash q_0BababB & \vdash Bbaq_1abB & \vdash Bbabq_2aB \\ \vdash Bq_1ababB & \vdash Bbabq_1bB & \vdash Bbaq_2baB \\ \vdash Bbq_1babB & \vdash Bbabaq_1B & \vdash Bbq_2abaB \end{array} \qquad \begin{array}{l} \vdash Bq_2babaB \\ \vdash q_2BbabaB \end{array}$$

Number of steps required with input string size n: $1 + n + 1 + n = 2n + 2$

2.3 Example
Copying a string: BuB becomes $BuBuB$ (u is a string of a's and b's)

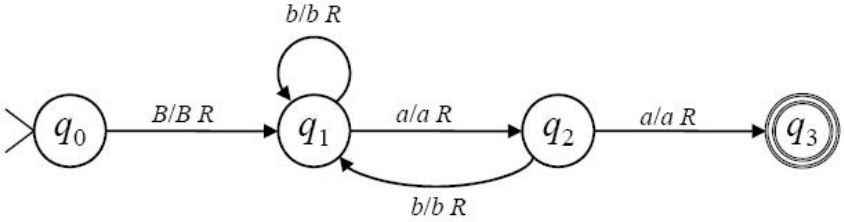


Number of steps required with input string size n: $O(n^2)$.

Exercise 8 Make a standard TM which moves an input string consisting of a's and b's one position to the right. What is the complexity of your TM?

Language accepting TMs additionally have a set $F \subseteq Q$ of *final* states. (They are sextuples.) A string is accepted if the computation halts in a final state (and does not terminate abnormally).

2.4 Example
A TM for $(a \cup b)^* aa (a \cup b)^*$.



Note that the TM assumes $\Sigma = \{a, b\}$.
Number of steps required with input string size n: n (worst case)

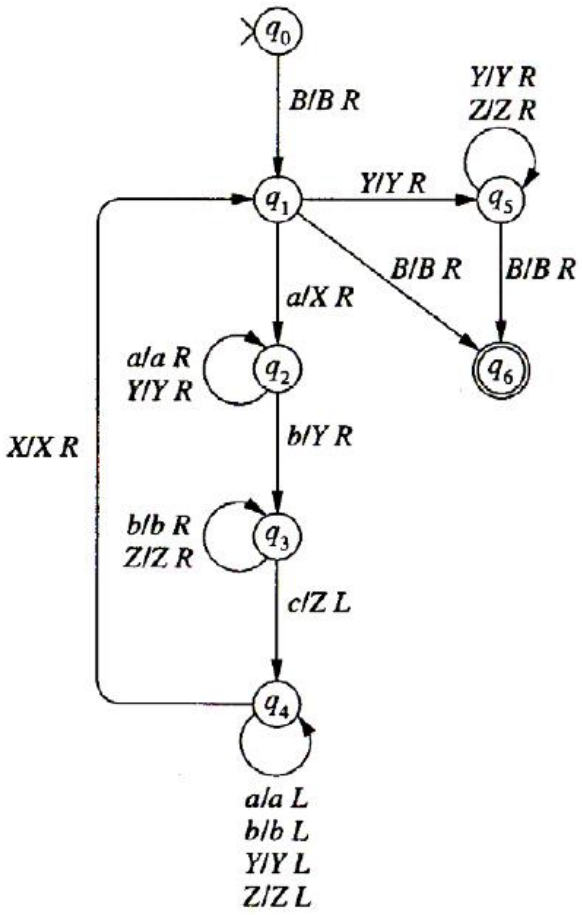


Figure 1: A TM for $\{a^i b^i c^i \mid i \geq 0\}$

2.5 Example
A TM for $\{a^i b^i c^i \mid i \geq 0\}$ (Figure 1).
Number of steps required with input string size $n = 3i$: $O(i \cdot 4i) = O(i^2) = O(n^2)$ (worst case)

Exercise 9 Make a standard TM which accepts the language $\{a^{2i} b^i \mid i \geq 0\}$. What is the complexity of your TM?

2.6 Example
A TM for palindromes over a and b (Figure 2).
Number of steps required with input string size n:

$$1 + \sum_{i=1}^{n} i = 1 + \frac{1}{2} \cdot (n+1) \cdot n \in O(n^2) \qquad \text{(worst case)}$$

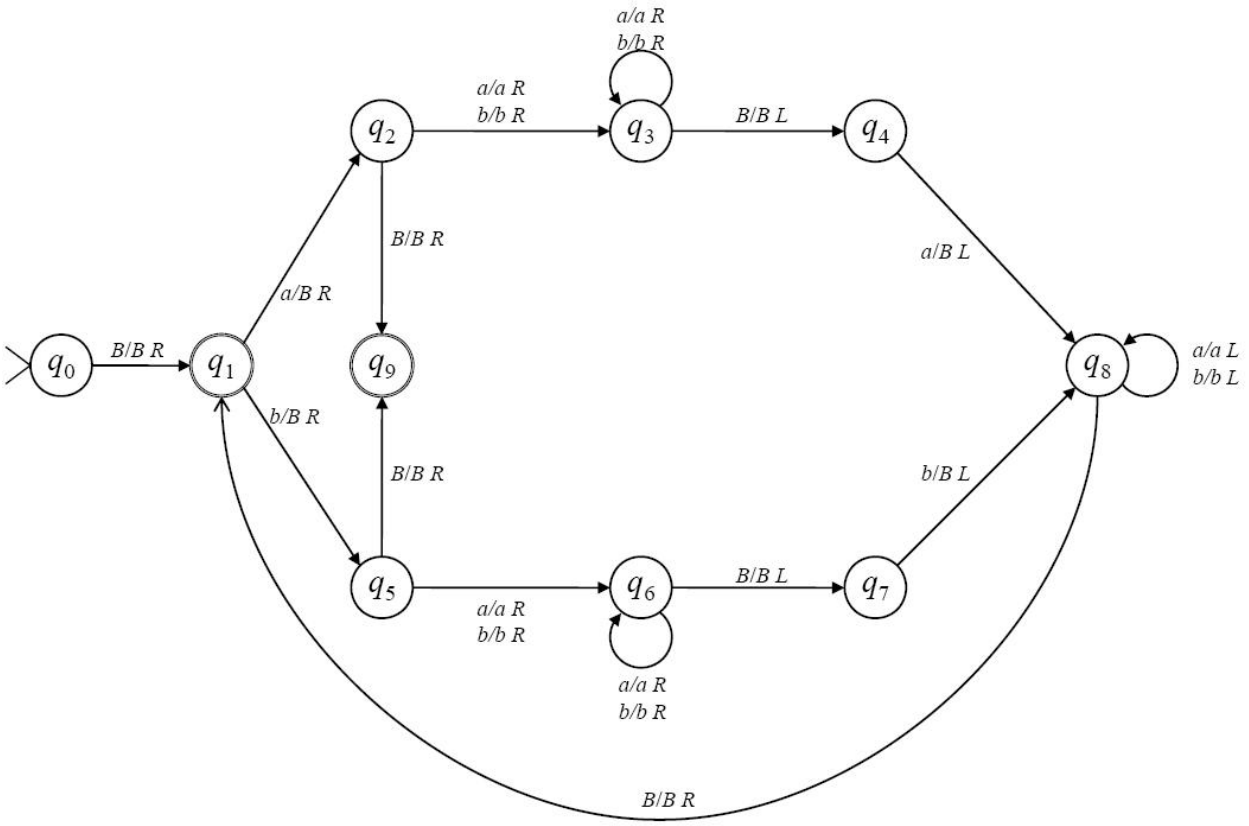


Figure 2: A TM for palindromes over a and b.

2.7 Definition
For any TM M, the *time complexity* of M is the function $tc_M : \mathbb{N} \to \mathbb{N}$ s.t. $tc_M(n)$ is the maximum number of transitions processed by a computation of M on input of length n.

2.8 Definition
A language L is *accepted in deterministic time* $f(n)$ if there is a single tape deterministic TM M for it with $tc_M(n) \in O(f(n))$.

Exercise 10 Design a single tape TM M for $\{a^i b^i \mid i \geq 0\}$ with $tc_M \in O(n \log_2(n))$. [Hint: On each pass, mark half of the a's and b's that have not been previously marked.]

2.9 Remark
Note, that worst-case behavior can happen when a string is *not* accepted. [E.g., straightforward TM to accept strings containing an a.]

3 Complexity under Turing Machine Variations

[4, Chapter 14.3 cont., 14.4 and recap Chapters 8.5, 8.6]

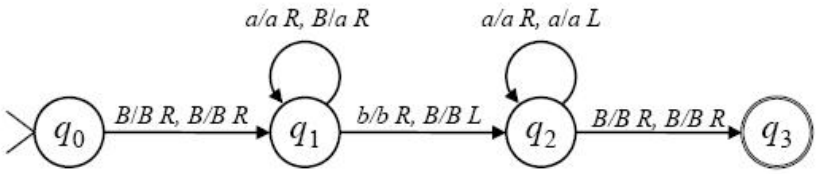
A *k-track TM* has one tape with k tracks. A single read-write head simultaneously reads the k symbols at the head position from all k tracks. We write the transitions as $\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}$. The input string is on track 1.

3.1 Theorem
If L is accepted by a k-track TM M then there is a standard TM M' which accepts L s.t. $tc_{M'}(n) = tc_M(n)$.

Proof: For $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, let $M' = (Q, \Sigma \times \{B\}^{k-1}, \Gamma^k, \delta', q_0, F)$ with transition function $\delta'(q_i, (x_1, \ldots, x_k)) = \delta(q_i, [x_1, \ldots, x_k])$. The number of transitions needed for a computation is unchanged. ■

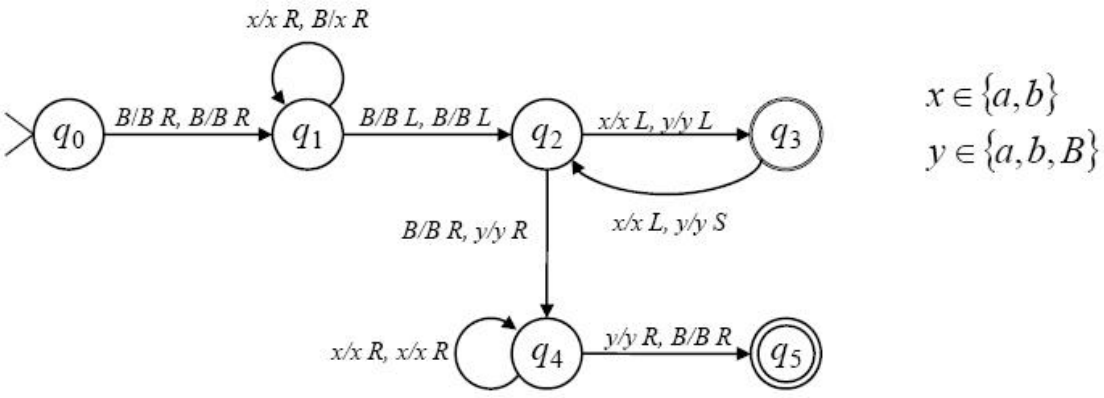
A *k-tape TM* has k tapes and k independent tape heads, which read simultaneously. A transition (i) changes the state, (ii) writes a symbol on each tape, and (iii) independently moves all tape heads. Transitions are written $\delta : (q_i, x_1, \ldots, x_k) \mapsto [q_j; y_1, d_1; \ldots; y_k, d_k]$ where $x_l, y_l \in \Gamma$ and $d_i \in \{L, R, S\}$ (S means head stays). Any head moving off the tape causes an abnormal termination. The input string is on tape 1.

3.2 Example
2-tape TM for $\{a^i b a^i \mid i \geq 0\}$.



Number of steps required with input string size n: $n + 2$

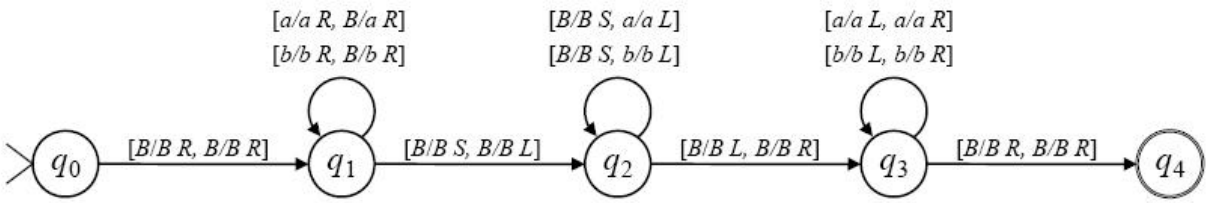
3.3 Example
2-tape TM for $\{uu \mid u \in \{a, b\}^\star\}$.



Number of steps required with input string size n: $\frac{5}{2}n + 4$

3.4 Example

2-tape TM accepting palindromes.



Number of steps required with input string size n: $3(n+1)+1$

3.5 Theorem

If L is accepted by a k-tape TM M then there is a standard TM N which accepts L s.t. $tc_N(n) = O(tc_M(n)^2)$.

Proof: (sketch) By Theorem 3.1 it suffices to construct an equivalent $2k+1$-track TM M' s.t. $tc_{M'}(n) \in O(tc_M(n)^2)$.

Simulation of the TM:

We show how to do this for $k = 2$ (but the argument generalizes).

Idea: tracks 1 and 3 maintain info on tapes 1 and 2 of M; tracks 2 and 4 have a single nonblank position indicating the position of the tape heads of M.

Initially: write # in track 5, position 1 and X in tracks 2 and 4, position 1.
States: 8-tuples of the form $[s, q_i, x_1, x_2, y_1, y_2, d_1, d_2]$, where $q_i \in Q$, $x_i, y_i \in \Sigma \cup \{U\}$, $d_i \in \{L, R, S, U\}$. $s \in \{f1, f2, p1\}$ represents the status of the simulation. U indicates an unknown item.

Let $\delta : (q_i, x_1, x_2) \mapsto [q_j; y_1, d_1; y_2, d_2]$ be the applicable transition of M.
M' start state: $[f1, q_i, U, U, U, U, U, U]$. The following actions simulate the transition of M:

1. $f1$ (find first symbol): M' moves to the right until X on track 2.
   Enter state $[f1, q_i, x_1, U, U, U, U, U]$, where x_1 is symbol on track 1 under x.
   M' returns to the position with # in track 5.
2. $f2$ (find second symbol): Same as above for recording symbol x_2 in track 3 under X in track 4.
   Enter state $[f2, q_i, x_1, x_2, U, U, U, U]$.
   Tape head returns to #.
3. Enter state $[p1, q_j, x_1, x_2, y_1, y_2, d_1, d_2]$, with q_j, y_1, y_2, d_1, d_2 obtained from $\delta(q_i, x_1, x_2)$.
4. $p1$ (print first symbol): move to X in track 2.
   Write symbol y_1 on track 1. Move X on track 2 in direction indicated by d_1.
   Tape head returns to #.
5. $p2$ (print second symbol): move to X in track 4.
   Write symbol y_2 on track 3. Move X on track 4 in direction indicated by d_2.
   Tape head returns to #.

6. Enter state $[f1, q_j, U, U, U, U, U, U]$.

If $\delta(q_i, x_1, x_2)$ is undefined, then simulation halts after step 2. $[f2, q_i, x_1, y_1, U, U, U, U]$ is accepting whenever q_i is accepting.

For each additional tape, add two tracks, and states obtain 3 more parameters. The simulation has 2 more actions (a find and a print for the new tape).

Complexity analysis:

Assume we simulate the t-th transition of M.
Heads of M are maximally at positions t.
Finds require maximum of $k \cdot 2t$ steps.
Prints require maximum of $k \cdot 2(t+1)$ steps.
Simulation of t-th transition requires maximum of $4kt + 2k + 2$ steps.

Thus

$$tc_{M'}(n) \leq 1 + \sum_{t=1}^{tc_M(n)} (4kt + 2k + 2) \in O(tc_M(n)^2).$$

■

Exercise 11 Let M be the TM from Example 3.2 and let M' (standard 1-track!) be constructed as in the proof of Theorem 3.5. Determine the number of states of M' which are 8-tuples.

Exercise 12 Let a *Random Access TM* (RATM) be a one-tape TM where transitions are of the form $\delta(q_i, x) = (q_j, y, d)$, where $d \in \mathbb{N}$. Such a transition is performed as usual, but the tape head is then moved to position d on the tape.
Give a sketch, how a RATM can be simulated by a standard TM.

Exercise 13 Do you think that the following is true?
For any RATM M there is a standard TM N such that $tc_N(n) = O(tc_M(n))$.
Justify your answer (no full formal proof needed).

4 Linear Speedup

[4, Chapter 14.5]

4.1 Theorem

Let M be a k-tape TM, $k > 1$, that accepts L with $tc_M(n) = f(n)$. Then, for any $c > 0$, there is a k-tape TM N that accepts L with $tc_N(n) \leq \lceil c \cdot f(n) \rceil + 2n + 3$.

4.2 Corollary

Let M be a 1-tape TM that accepts L with $tc_M(n) = f(n)$. Then, for any $c > 0$, there is a 2-tape TM N that accepts L with $tc_N(n) \leq \lceil c \cdot f(n) \rceil + 2n + 3$.

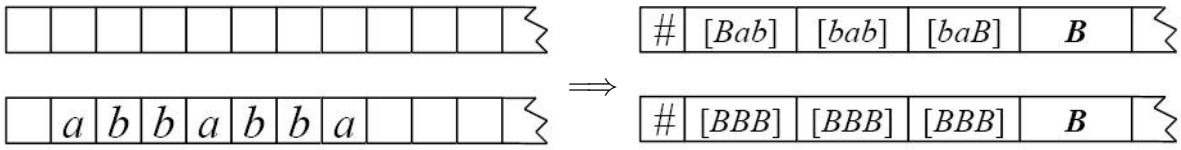
Proof: The 1-tape TM can be understood as a 2-tape TM where tape 2 is not used. Then apply Theorem 4.1. ■

Exercise 14 Assume a language L is accepted by a 2-tape TM M with $tc_M(n) = \sqrt{n}$. Do you think it is possible to design a standard TM N for L with $tc_N(n) \in O(n)$? Justify your answer.

Proof sketch/idea for Theorem 4.1:
Exemplified using Example 3.4.
$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$.

Simulation of the TM:

N input alphabet: Σ.
N tape alphabet: $\Gamma \cup \Gamma^m \cup \{\#, ?\}$
Initialization of N by example.



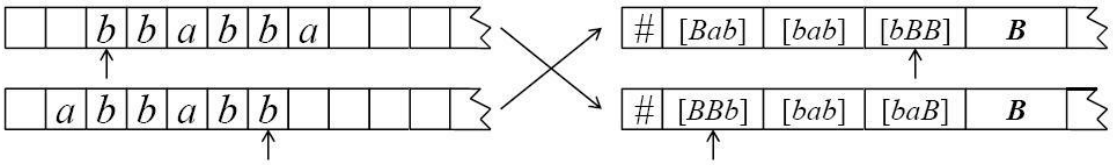
A state of N consists of:

- the state of M
- for $i = 1, \ldots, k$, the m-tuple currently scanned on tape i of N and the m-tuples to the immediate right and left
- a k-tuple $[i_1, \ldots, i_k]$, where i_j is the position of the symbol on tape j being scanned by M in the m-tuple being scanned by N.

After initialization, enter state

$$(q_0; ?, [BBB], ?; ?, [Bab], ?; [1, 1])$$

(? is a placeholder, filled in by subsequent movements).
Idea: Six transitions of N simulate m transitions of M (m depends on c).
Simulated, compressed configurations:



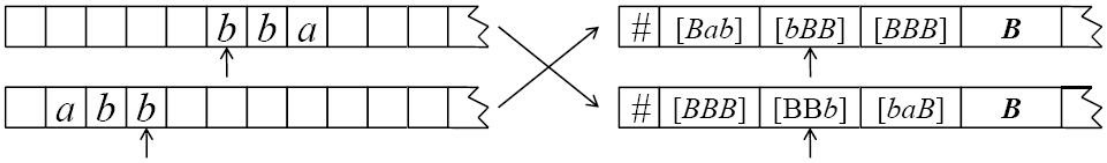
State:

$$(q3; ?, [BBb], ?; ?, [bBB], ?; [3, 1])$$

Six transitions are performed:

1. move left, record tuples in the state
2. move two to the right, record tuples in the state
3. move one to the left
   State: $(q_3, \#, [BBb], [bab]; [bab], [bBB], [BBB]; [3, 1])$

4. and
5. rewrite tapes to match configuration of M after three transitions.



State: $(q_3, ?, [BBb], ?; ?, [bBB], ?; [3, 1])$

Complexity analysis:

Initialization requires $2n + 3$ transitions.
For simulation, 6 moves of N simulate m moves of M.
With $m \geq \frac{6}{c}$ we obtain

$$tc_N(n) = \left\lceil \frac{6}{m} f(n) \right\rceil + 2n + 3 \leq \lceil c \cdot f(n) \rceil + 2n + 3$$

as desired.

4.3 Remark
In Theorem 4.1, the speedup is obtained by using a larger tape alphabet and by vastly increasing the number of states.

5 Properties of Time Complexity

[4, Chapter 14.6]

5.1 Theorem
Let f be a total computable function. Then there is a language L such that tc_M is not bounded by f for any TM M that accepts L.

Proof sketch: Let u be an injective function which assigns to every TM M with $\Sigma = \{0, 1\}$ a string $u(M)$ over $\{0, 1\}$. [Each TM can be described by a finite string of characters; then consider a bit encoding of the characters.]
Let $u_1, u_2, u_3, \ldots$ be an enumeration of all strings over $\{0, 1\}$. If $u(M) = u_i$ for some M, then set $M_i = M$, otherwise set M_i to be the one-state TM with no transitions.
This needs to be done, such that there is a TM N which, on input any u_i, can simulate M_i.

$$L = \{u_i \mid M_i \text{ does not accept } u_i \text{ in } f(\text{length}(u_i)) \text{ or fewer moves}\}.$$

L is recursive. [Input some u_i. Determine length(u_i). Compute f(length(u_i)). Simulate M_i on u_i until M_i either halts or completes f(length(u_i)) transitions, whichever comes first. u_i is accepted if either M_i halted without accepting u_i or M_i did not halt in the first f(length(u_i)) transitions. Otherwise, u_i is rejected.]

Let M be a TM accepting L. Then $M = M_j$ for some j. $M = M_j$ accepts u_j iff M_j halts on input u_j without accepting u_j in $f(\text{length}(u_j))$ or fewer steps or M_j does not halt in the first $f(\text{length}(u_j))$ steps.
Hence, if M accepts u_j then it needs more than $f(\text{length}(u_j))$ steps.
If M does not accept u_j, then it also cannot stop in $f(\text{length}(u_j))$ or fewer steps, since then it would in fact accept u_j. ■

5.2 Theorem
There is a language L such that, for any TM M that accepts L, there is a TM N that accepts L with $tc_N(n) \in O(\log_2(tc_M(n)))$.

Proof: skipped ■

Exercise 15 Is the following true or false? Prove your claim.
For the language L from Theorem 5.2, the following holds: If there is a TM M that accepts L with $tc_M(n) \in O(2^n)$, then there is a TM N that accepts L with $tc_N(n) \in O(n)$.

Exercise 16 Is the following true or false? Prove your claim.
Let M be a 5-track TM which accepts a language L. Then there is a 5-tape TM N that accepts L with

$$tc_N(n) \leq \frac{7 + 7n + tc_M(n)}{2}.$$

6 Simulation of Computer Computations

[4, Chapter 14.7]

Is the TM complexity model adequate?
Assume a computer with the following parameters.

- finite memory divided into word-size chunks
- fixed word length, m_w bits each
- each word has a fixed numeric address
- finite set of instructions
- each instruction
  - fits in a single word
  - has maximum t operands (addresses used in operation)
  - can do one of
    - move data
    - perform arithmetic or Boolean calculations
    - adjust the program flow
    - allocate additional memory (maximum of m_a words each time)
  - can change at most t words in the memory

Now simulate in $5+t$-tape TM. Tapes are:

- Input tape (divided into word chunks)
- Memory counter (address of next free word on tape 1)
- Program counter (location of next instruction to be exectuted)
- Input counter (location of beginning of input and location of next word to be read)
- Work tape
- t Register tapes

Simulation of k-th instruction:

1. load operand data onto the register tapes (max t words)
2. perform indicated operation (one)
3. store results as requried (stores max t words or allocates m_a words of memory)

Operation: finite number of instructions with at most t operands. Maximum number of transitions needed shall be t_0 (max exists and is $< \infty$)
Load and Store:
m_p number of bits used to store input
m_i number of bits used to store instructions
m_k total memory allocated by TM before instruction k

$$m_k \leq m_p + m_i + k \cdot m_a$$

Any address can be located in m_k transitions.
Upper bounds:

find instruction	m_k
load operands	$t \cdot m_k$
return register tape heads	$t \cdot m_k$
perform operation	t_0
store information	$t \cdot m_{k+1}$
return register tape heads	$t \cdot m_{k+1}$

upper bound for k-th instruction:

$$(2t+1)m_k + 2tm_{k+1} + t_0 \leq (4t+1)m_p + (4t+1)m_i + 2tm_a + t_0 + (4t+1)km_a$$

If computer requires $f(n)$ steps on input length $m_p = n$, then simulation requires:

$$\begin{aligned}
&\sum_{k=1}^{f(n)}((4t+1)n + (4t+1)m_i + 2tm_a + t_0 + (4t+1)km_a) \\
&\qquad = f(n)((4t+1)n + (4t+1)m_i + 2tm_a + t_0) + \sum_{k=1}^{f(n)}(4t+1)km_a \\
&\qquad = f(n)((4t+1)n + (4t+1)m_i + 2tm_a + t_0) + (4t+1)m_a \sum_{k=1}^{f(n)} k \\
&\qquad \in O(\max\{nf(n), f(n)^2\})
\end{aligned}$$

In particular:
If an algorithm runs in polynomial time on a computer, then it can be simulated on a TM in polynomial time. [For $f(n) \in O(n^r)$, we have $nf(n) \in O(n^{r+1})$ and $f(n)^2 \in O(n^{2r})$]

Exercise 17 Can we conclude the following from the observations in this section?

- *If an algorithm runs in exponential time on a computer, then it can be simulated on a TM in exponential time.*
- *If an algorithm runs in linear time on a computer, then it can be simulated on a TM in quadratic time.*

Justify your answer.

7 Computability

See slides.

8 PTime

[[4, Chapter 15.6] and some bits and pieces]

8.1 Definition
A language L is *decidable in polynomial time* if there is a standard TM M that accepts L with $tc_M(n) \in O(n^r)$, where $r \in \mathbb{N}$ is independent of n. $\mathcal{P}$ ($PTime$) is the *complexity class* of all such languages. [$\mathcal{P}$ is the *set* of all such languages.]

Exercise 18 Show that $\mathcal{P}$ is closed under language complement.

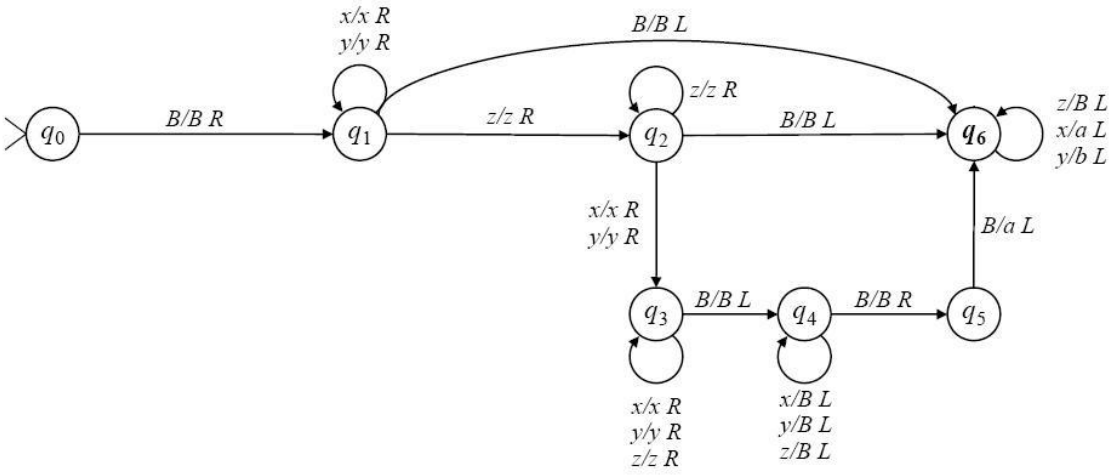
8.2 The Polynomial Time Church-Turing Thesis
A decision problem can be solved in polynomial time by using a reasonable sequential model of computation if and only if it can be solved in polynomial time by a Turing Machine.

We have seen that $\mathcal{P}$ is independent of the computation paradigm used:

- standard TMs
- k-track TM
- k-tape TM
- "realistic" computer

8.3 Definition
Let L, Q, be languages over alphabets Σ_1 and Σ_2, respectively. L is *reducible* (*in polynomial time*) to Q if there is a polynomial-time computable function $r : \Sigma_1^* \to \Sigma_2^*$ such that $w \in L$ if and only if $r(w) \in Q$.

8.4 Example
The TM



reduces $L = \{x^i y^i z^k \mid i, k \geq 0\}$ to $Q = \{a^i b^i \mid i \geq 0\}$.
Time complexity: $O(n)$.

Exercise 19 Construct a TM which reduces

$$\{a^i b^i a^i \mid i \geq 0\} \text{ to } \{c^i d^i \mid i \geq 0\}.$$

8.5 Theorem
Let L be reducible to Q in polynomial time and let $Q \in \mathcal{P}$. Then $L \in \mathcal{P}$.

Proof: TM for reduction: R.
TM deciding Q: M.
R on input w generates $r(w)$ as input to M.
$\text{length}(r(w)) \leq \max\{\text{length}(w), tc_R(\text{length}(w))\}$.
If $tc_R \in O(n^s)$ and $tc_M \in O(n^t)$, then

$$tc_R(n) + tc_M(\max\{n, tc_R(n)\}) \in O(n^s) + O(\max\{O(n^t), O((n^s)^t)\}) = O(n^{st}).$$

■

8.6 Definition
A language (problem) L is

- $\mathcal{P}$-*hard*, if every language in $\mathcal{P}$ is reducible to L in polynomial time.
- $\mathcal{P}$-*complete*, if L is $\mathcal{P}$-hard and $L \in \mathcal{P}$.

8.7 Remark
Definition 8.6 (hardness and completeness) are used likewise for other complexity classes. Thereby, reducibility is always considered to be polynomial time.

8.8 Remark
In principle, you could use any decision problem for defining a complexity class. E.g., if the *POPI-problem* is to find out, if n potachls fit into a pistochl of size n, then a problem/language L is

- *in POPI* if L is reducible to the POPI-problem (in polynomial time),
- *POPI-hard* if the POPI-problem is reducible to L (in polynomial time),
- *POPI-complete* if it is both in POPI and POPI-hard.

Obvious questions:

- Which complexity classes are interesting or useful?
- How do they relate to each other?

In this class, we mainly talk about two complexity classes: $\mathcal{P}$, and $\mathcal{NP}$ (soon).

Exercise 20 By Theorem 5.1, there are problems which are not in $\mathcal{P}$. Go to

`http://qwiki.stanford.edu/wiki/Complexity_Zoo`

and do the following.

1. Find the names of 4 complexity classes which contain $\mathcal{P}$.
2. Find a $\mathcal{P}$-hard problem and describe it briefly in general, but understandable, terms. (You may have to use other sources for background understanding.)

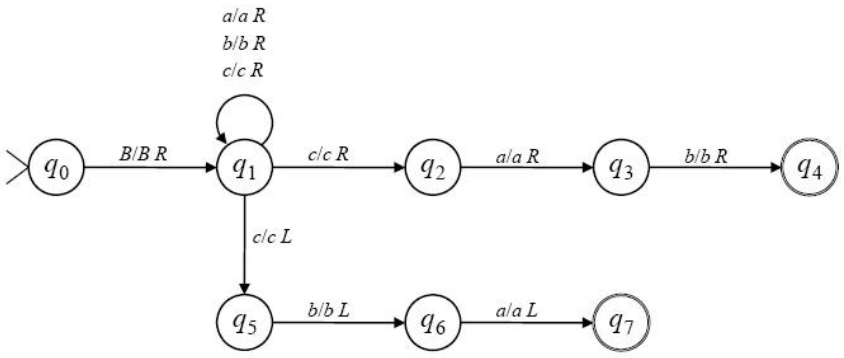
9 Nondeterministic Turing Machines and Time Complexity

[4, Chapters 15.1, 15.2 cont., some of Chapter 7.1, and recap Chapter 8.7]

- A nondeterministic (ND) TM may specify any finite number of transitions for a given configuration.
- Transitions are defined by a function from $Q \times \Gamma$ to the set of finite subsets of $Q \times \Gamma \times \{L, R\}$.
- A computation arbitrarily chooses one of the possible transitions. Input is accepted if there is at least one computation terminating in an accepting state.

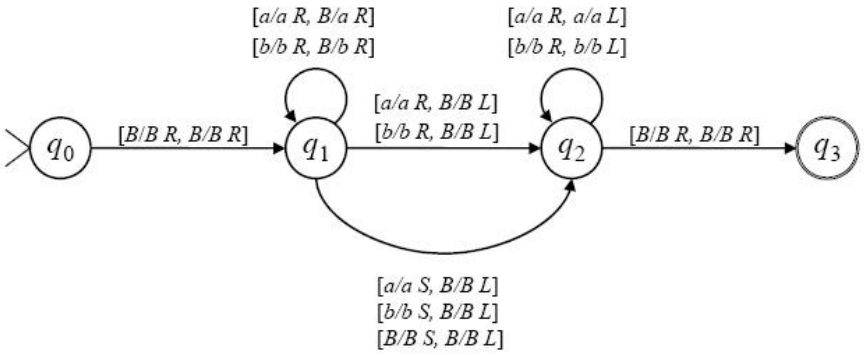
9.1 Definition
Time complexity $tc_M : \mathbb{N} \to \mathbb{N}$ of an ND TM M is defined s.t. $tc_M(n)$ is the maximum number of transitions processed by input of length n.

9.2 Example
Accept strings with a c preceeded or followed by ab.



Complexity: $O(n)$

9.3 Example
2-tape ND palindrome finder.



Complexity: $n + 2$ if n is odd, $n + 3$ if n is even.

Exercise 21 Give a nondeterministic two-tape TM for $\{uu \mid u \in \{a, b\}^*\}$ which is quicker than the TM from Example 3.3.

9.4 Definition
A language L is accepted *in nondeterministic polynomial time* if there is an ND TM M that accepts L with $tc_M \in O(n^r)$, where $r \in \mathbb{N}$ is independent of n. $\mathcal{NP}$ is the *complexity class* of all such languages.

9.5 Remark
$\mathcal{P} \subseteq \mathcal{NP}$. It is currently not known if $\mathcal{P} = \mathcal{NP}$.

9.6 Theorem
Let L be accepted by a one-tape ND TM M. Then L is accepted by a deterministic TM M' with $tc_{M'}(n) \in O(tc_M(n)c^{tc_M(n)})$, where c is the maximum number of transitions for any state, symbol pair of M.

Proof sketch: Simulation idea: Use 3-tape TM M'. Tape 1 holds input. Tape 2 is used for simulating the tape of M. Tape 3 holds sequences $(m_1, \ldots, m_k)$ $(1 \leq m_i \leq c)$, which encode computations of M: m_i indicates that, from the (maximally) c choices M has in performing the i-th transition, the m_i-th choice is selected.
M is simulated as follows:

1. generate a $(m_1, \ldots, m_k)$
2. simulate M according to $(m_1, \ldots, m_k)$
3. if input is not accepted, continue with step 1.

Worst case: $c^{tc_M(n)}$ sequences need to be examined. Simulation of a single computation needs maximally $O(tc_M(n))$ transitions. Thus, $tc_{M'}(n) \in O(tc_M(n)c^{tc_M(n)})$. ■

Exercise 22 Make a (deterministic) pseudo-code algorithm for an exhaustive search on a tree (i.e., if the sought element is not found, the whole tree should be traversed).

Exercise 23 Make a non-deterministic pseudo-code algorithm for an exhaustive search on a tree.

9.7 Definition
co-$\mathcal{NP} = \{\overline{L} \mid L \in \mathcal{NP}\}$ and co-$\mathcal{P} = \{\overline{L} \mid L \in \mathcal{P}\}$, where $\overline{L}$ denotes the complement of L. It is currently not known if $\mathcal{NP} =$ co-$\mathcal{NP}$.

9.8 Theorem
If $\mathcal{NP} \neq$ co-$\mathcal{NP}$, then $\mathcal{P} \neq \mathcal{NP}$.

Proof: Proof by contraposition:
If $\mathcal{P} = \mathcal{NP}$, then by Exercise 18 we have

$$\mathcal{NP} = \mathcal{P} = \text{co-}\mathcal{P} = \text{co-}\mathcal{NP}.$$

■

9.9 Theorem
If there is an $\mathcal{NP}$-complete language L with $\overline{L} \in \mathcal{NP}$, then $\mathcal{NP} =$ co-$\mathcal{NP}$.

Proof: Assume L is a language as stated.
Let $Q \in \mathcal{NP}$. Then Q is reducible to L in polynomial time. This reduction is also a reduction of $\overline{Q}$ to $\overline{L}$.
Combining the TM which performs the reduction with the TM which accepts $\overline{L}$ results in an ND TM that accepts $\overline{Q}$ in polynomial time. Thus $Q \in$ co-$\mathcal{NP}$.
This shows $\mathcal{NP} \subseteq$ co-$\mathcal{NP}$. The inclusion co-$\mathcal{NP} \subseteq \mathcal{NP}$ follows by symmetry. ■

9.10 Theorem
Let Q be an $\mathcal{NP}$-complete language. If Q is reducible to L in polynomial time, then L is $\mathcal{NP}$-hard.

Proof: If $R \in \mathcal{NP}$, then R is reducible in polynomial time to Q, which in turn is reducible in polynomial time to L. By composition, R is reducible in polynomial time to L. ■

9.11 Remark
When moving from languages to decision problems, the representation of numbers may make a difference: Conversion from binary to unary representation is in $O(2^n)$.
Thus, if a problem can be solved in polynomial time with unary input representation, it may not be solvable in polynomial time with binary input representation.
Most reasonable representations of a problem differ only polynomially in length, but not so unary number encoding.
Thus, in complexity analysis, numbers are always assumed to be represented in binary.

9.12 Definition
A decision problem with a polynomial solution using unary number representation, but no polynomial solution using binary representation, is called *pseudo-polynomial*.

Exercise 24 Let L be the language of all strings over $\{a, b\}$ that can be divided into two strings (not necessarily the same length) such that (1) both strings have the same number of b's and (2) both strings start and end with a. E.g., *abbaababaa* is in L because it can be divided into *abba* and *ababaa*, both of which have 2 b's and both of which start and end in a. The string *bbabba* is not in L.
Give a 2-tape, single track ND TM that accepts L. Explain your TM in words.

10 SAT is $\mathcal{NP}$-Complete

[4, Chapter 15.8]

Let V be a set of *Boolean variables*.

10.1 Definition
An *atomic formula* is a Boolean Variable.
(*Well-formed*) *formulas* are defined as follows.

1. All atomic formulas are formulas.
2. For every formula F, $\neg F$ is a formula, called the *negation* of F.
3. For all formulas F and G, also $(F \vee G)$ and $(F \wedge G)$ are formulas, called the *disjunction* and the *conjunction* of F and G, respectively.
4. Nothing else is a formula.

10.2 Definition
$\mathbb{T} = \{0, 1\}$ – the set of *truth values*: *false*, and *true*, respectively.
An *assignment* is a function $\mathcal{A} : \mathbf{D} \to \mathbb{T}$, where $\mathbf{D}$ is a set of atomic formulas.
Assignments extend to formulas, via the following *truth tables*.

$\mathcal{A}(F)$	$\mathcal{A}(G)$	$\mathcal{A}(F \wedge G)$
0	0	0
0	1	0
1	0	0
1	1	1

$\mathcal{A}(F)$	$\mathcal{A}(G)$	$\mathcal{A}(F \vee G)$
0	0	0
0	1	1
1	0	1
1	1	1

$\mathcal{A}(F)$	$\mathcal{A}(\neg F)$
0	1
1	0

A formula F is called *satisfiable* if there exists an assignment $\mathcal{A}$ with $\mathcal{A}(F) = 1$. $\mathcal{A}$ is called a *model* of F in this case, and we write $\mathcal{A} \models F$.

10.3 Example
Determining the truth values of formulas using truth tables:

$\mathcal{A}(B)$	$\mathcal{A}(F)$	$\mathcal{A}(I)$	$\mathcal{A}(B \wedge F)$	$\mathcal{A}(\neg(B \wedge F))$	$\mathcal{A}(\neg I)$	$\mathcal{A}(\neg(B \wedge F) \vee \neg I)$
0	0	0	0	1	1	1
0	0	1	0	1	0	1
0	1	0	0	1	1	1
0	1	1	0	1	0	1
1	0	0	0	1	1	1
1	0	1	0	1	0	1
1	1	0	1	0	1	1
1	1	1	1	0	0	0

Exercise 25 Make the truth table for the formula $\neg(I \vee \neg B) \vee \neg F$.

Exercise 26 Give a formula F, containing only the Boolean variables A, B, and C, such that F has the following truth table.

$\mathcal{A}(A)$	$\mathcal{A}(B)$	$\mathcal{A}(C)$	$\mathcal{A}(F)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

10.4 Definition
Formulas F and G are (*semantically*) *equivalent* (written $F \equiv G$) if for every assignment $\mathcal{A}$, $\mathcal{A}(F) = \mathcal{A}(G)$.

10.5 Theorem
The following hold for all formulas F, G, and H.

$F \wedge G \equiv G \wedge F$	$F \vee G \equiv G \vee F$	Commutativity
$(F \wedge G) \wedge H \equiv F \wedge (G \wedge H)$	$(F \vee G) \vee H \equiv F \vee (G \vee H)$	Associativity
$F \wedge (G \vee H) \equiv (F \wedge G) \vee (F \wedge H)$	$F \vee (G \wedge H) \equiv (F \vee G) \wedge (F \vee H)$	Distributivity
$\neg\neg F \equiv F$		Double Negation
$\neg(F \wedge G) \equiv \neg F \vee \neg G$	$\neg(F \vee G) \equiv \neg F \wedge \neg G$	de Morgan's Laws

Proof: Straightforward using truth tables. ■

10.6 Definition
A *literal* is an atomic formula (a *positive* literal) or the negation of an atomic formula (a *negative* literal). A *clause* is a disjunction of literals.
A formula F is in *conjunctive normal form* (CNF) if it is a conjunction of clauses, i.e., if

$$F = \left(\bigwedge_{i=1}^{n} \left(\bigvee_{j=1}^{m_i} L_{i,j} \right) \right),$$

where the $L_{i,j}$ are literals.

10.7 Theorem
For every formula F there is a formula $F_1 \equiv F$ in CNF.

Proof: skipped ■

Exercise 27 Transform $\neg((A \vee B) \wedge (C \vee D) \wedge (E \vee F))$ into CNF.

Exercise 28 Give an informal, but plausible, argument, why a naive algorithm for converting formulas into CNFs is not in $\mathcal{P}$. [Don't use TMs.]

10.8 Definition
The *Satisfiability Problem* (SAT) is the problem of deciding if a formula in CNF is satisfiable.

10.9 Theorem (Cook's Theorem)
SAT is $\mathcal{NP}$-complete.

Proof: [Slideset 2: Proof of Cook's Theorem] ■

10.10 Remark
SAT is sometimes stated without the requirement that the formula is in CNF – this is equivalent.

10.11 Proposition
For any formula F, there is an equivalent formula which contains only $\wedge$, $\vee$, and literals. (Called a *negation normal form* (*NNF*) of F.)

Proof: Apply de Morgan's laws exhaustively. ■

Exercise 29 Give an informal, but plausible, argument that conversion of a formula into NNF is in $\mathcal{P}$.

10.12 Definition
Two formulas F and G are *equisatisfiable* if the following holds: F has a model if and only if G has a model.

10.13 Proposition
For all formulas F_i $(i = 1, 2, 3)$, $F_1 \vee (F_2 \wedge F_3)$ and $(F_1 \vee E) \wedge (\neg E \vee (F_2 \wedge F_3))$ are equisatisfiable (where E is a propositional variable not occurring in F_1, F_2, F_3).

Proof: skipped ■

Exercise 30 Use the idea from Proposition 10.13 to sketch a polynomial-time algorithm which converts any formula F into an equisatisfiable formula in CNF. [Hint: First convert into NNF.]

Exercise 31 Give an informal, but plausible, argument, that the problem "*Decide if a formula is satisfiable*" is $\mathcal{NP}$-complete. [Use Cook's Theorem and Exercise 30.]

11 If $\mathcal{P} \neq \mathcal{NP}$ ...

[A mix, mainly from [1, Chapter 7], with some from [4, Chapter 17] and other sources.]

11.1 Problems "between" $\mathcal{P}$ and $\mathcal{NP}$

$\mathcal{NPC}$ consists of all $\mathcal{NP}$-complete languages.
If $\mathcal{P} \subsetneq \mathcal{NP}$, is $\mathcal{NPI} = \mathcal{NP} \setminus (\mathcal{P} \cup \mathcal{NPC}) \neq \emptyset$?

11.1 Theorem
Let B be a recursive language such that $B \notin \mathcal{P}$. Then there exists $D \in P$ s.t. $A = D \cap B \notin \mathcal{P}$, A is (polytime) reducible to B but B is not (polytime) reducible to A.

Exercise 32 Assumed $\mathcal{P} \neq \mathcal{NP}$, why does Theorem 11.1 show that $\mathcal{NPI} \neq \emptyset$?

Iteratively reapplying the argument from Exercise 32 yields an infinite collection of distinct complexity classes "between" $\mathcal{P}$ and $\mathcal{NP}$.
Are there any "natural" candidates for problems in $\mathcal{NPI}$? Perhaps the following.

- **Graph isomorphism:** Given two graphs $G = (V, E)$ and $G' = (V, E')$, is there a bijection $f : V \to V$ such that $(u, v) \in E$ iff $(f(u), f(v)) \in E'$?

The following was for a long time believed to be a candidate for a probloem in $\mathcal{NPI}$, but in 2004 it was shown that it is in $\mathcal{P}$.

- **Composite numbers (the primality problem):** Given $k \in \mathbb{N}$, are there $1 < n, m \in \mathbb{N}$ s.t. $k = m \cdot n$?

11.2 The Polynomial Hierarchy

11.2 Definition
An *oracle TM* (OTM) is a standard TM with an additional *oracle tape* with read-write *oracle head*. It has two additional distinguished states: the *oracle consultation state* and the *resume-computation state*. Also, an *oracle function* $g : \Sigma^* \to \Sigma^*$ is given.
Computation is as for a 2-tape TM, except in the oracle state: If y is on the oracle tape (right of the first blank), then it is rewritten to $g(y)$ (with rest blank) in one step, and the state is changed to the resume state.

Let C and D be two complexity classes (sets of languages). Denote by C^D the class of all languages which are accepted by an OTM of complexity C, where computation of the oracle function has complexity D.

11.3 Remark
$\mathcal{P}^{\mathcal{P}} = \mathcal{P}$ [The one-step oracle consultation can be performed using a TM which runs in polynomial time. Note that there can be at most polynomially many such consultations.]

Exercise 33 Show: $\mathcal{P}^{\mathcal{NP}}$ contains all languages which are (polynomial-time) reducible to a language in $\mathcal{NP}$.

11.4 Definition
The *polynomial hierarchy*:

$$\Sigma_0^p = \Pi_0^p = \Delta_0^p = \mathcal{P}$$

and for all $k \geq 0$

$$\begin{aligned}\Delta_{k+1}^p &= \mathcal{P}^{\Sigma_k^p} \\ \Sigma_{k+1}^p &= \mathcal{NP}^{\Sigma_k^p} \\ \Pi_{k+1}^p &= \text{co-}\Sigma_{k+1}^p\end{aligned}$$

PH is the union of all classes in the polynomial hierarchy.

Exercise 34 Show $\Sigma_1^p = \mathcal{NP}$.

11.5 Remark
$\Pi_1^p = \text{co-}\mathcal{NP}^{\mathcal{P}} = \text{co-}\mathcal{NP}$
$\Delta_2^p = \mathcal{P}^{\Sigma_1^p} = \mathcal{P}^{\mathcal{NP}}$

11.6 Remark
$\Sigma_i^p \subseteq \Delta_{i+1}^p \subseteq \Sigma_{i+1}^p$
$\Pi_i^p \subseteq \Delta_{i+1}^p \subseteq \Pi_{i+1}^p$
$\Sigma_i^p = \text{co-}\Pi_i^p$
It is not known if the inclusions are proper. If any Σ_k^p equals Σ_{k+1}^p or Π_k^p, then the hierarchy collapses above k. In particular, if $\mathcal{P} = \mathcal{NP}$, then $\mathcal{P} = \text{PH}$.

The following problem may be in $\Sigma_2^p = \mathcal{NP}^{\mathcal{NP}}$:
Maximum equivalent expression: Given a formula F and $k \in \mathbb{N}$, is there $F_1 \equiv F$ with k or fewer occurrences of literals?
[$\mathcal{NP}$-hardness: because SAT reduces to it.
In Σ_2^p: Use SAT (non-CNF version) as oracle. The OTM first guesses F_1, then consults the oracle.]

Exercise 35 Spell out in more detail, how SAT reduces to *maximum equivalent expression*.

12 Beyond $\mathcal{NP}$

[Mainly from [4, Chapter 17], plus some from [1, Chapter 7] and other sources.]

Space complexity: use modified k-tape TM (*off-line TM*) with additional read-only *input tape*, and additional write-only *output tape*. The latter is not needed for language recognition tasks.

12.1 Definition
The *space complexity* of a TM M is the function $sc_M : \mathbb{N} \to \mathbb{N}$ s.t. $sc_M(n)$ is the maximum number of tape squares read on any work tape by a computation of M when initiated with an input string of length n. (For an ND TM, take the maximum over every possible computation as usual.)

12.2 Example
3-tape palindrome recognizer M with $sc_M(n) = O(\log_2(n))$.
Idea: Use work tapes to hold numbers (binary encoding). They are used as counters for identifying and comparing the i-th element from the left with the i-th element from the right, until a mismatch is found (or the palindrome is accepted).

12.3 Remark
Palindrome recognition is in the complexity class LOGSPACE.

12.4 Theorem
For any TM M, $sc_M(n) \leq tc_M(n) + 1$.

12.5 Definition
An off-line TM is said to be $s(n)$ *space-bounded* if the maximum number of tape squares used on a work tape with input of length n is at most $\max\{1, s(n)\}$. (This can also be used with non-terminating TMs.)

12.6 Theorem (Savitch's Theorem)
M a 2-tape ND TM with space bound $s(n) \geq \log_2(n)$ which accepts L. Then L is accepted by a deterministic TM with space bound $O(s(n)^2)$.

12.7 Corollary
$\mathcal{P}$-Space = $\mathcal{NP}$-Space

Proof: Obviously, $\mathcal{P}$-Space $\subseteq \mathcal{NP}$-Space.
If $L \in \mathcal{NP}$-Space, then it is accepted by an ND TM with polynomial space bound $p(n)$. Then by Savitch's Theorem, L is accpeted by a deterministic TM with space bound $O(p(n)^2)$. ■

12.8 Definition
EXPTIME is the complexity class of problems solvable by a (deterministic) TM in $O\left(2^{g(n)}\right)$ time, where g is a polynomial. NEXPTIME is the corresponding ND class. 2-EXPTIME/N2-EXPTIME are defined similarly with time bound $O(2^{2^{g(n)}})$. (Similarly n-EXPTIME – the *exponential hierarchy.*) (EXPSPACE is the corresponding space complexity class.)

Exercise 36 Show, that EXPSPACE=NEXPSPACE.

12.9 Remark
It is known that LOGSPACE $\subseteq$ NLOGSPACE $\subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq$ PH $\subseteq \mathcal{P}$-Space = $\mathcal{NP}$-Space $\subseteq$ EXPTIME $\subseteq$ NEXPTIME $\subseteq$ EXPSPACE $\subseteq$ 2-EXPTIME.
Also known:

- LOGSPACE $\subsetneq \mathcal{P}$-Space
- $\mathcal{P} \subsetneq$ EXPTIME
- $\mathcal{NP} \subsetneq$ NEXPTIME
- $\mathcal{P}$-Space $\subsetneq$ EXPSPACE
- If $\mathcal{P} = \mathcal{NP}$, then EXPTIME = NEXPTIME
- If $\mathcal{P} = \mathcal{NP}$, then $\mathcal{P}$ = PH

12.10 Remark
The Web Ontology Language OWL-DL (see [2]) is N2-EXPTIME-complete. The description logic $\mathcal{ALC}$ (see [3]) is EXPTIME-complete.

Exercise 37 Show: If $\mathcal{NPI} = \emptyset$, then $\mathcal{NP} \neq$ EXPTIME.

12.11 Example
$\mathcal{P}$-Space-complete problems:

- **Regular expression non-universality:** Given a regular expression α over a finite alphabet Σ, is the set represented by α different from Σ^*?
- **Linear space acceptance:** Given a linear space-bounded TM M and a finite string x over its input alphabet, does M accept x?

13 Excursus: Is $\mathcal{P} = \mathcal{NP}$?

[mainly from memory]

[blackboard]

Exercise 38 Show $|\mathbb{R}| = |\{(1, x) : x \in \mathbb{R}\} \cup \{(2, x) : x \in \mathbb{R}\}|$.

Exercise 39 Show, using a diagonalization argument, that the power set of $\mathbb{N}$ is of higher cardinality than $\mathbb{N}$. [Hint: Consider only infinite subsets of $\mathbb{N}$.]

References

[1] M. R. Garey and D. S. Johnson. *Computers and Intractability*. Freeman, 1979.

[2] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph, editors. *OWL 2 Web Ontology Language: Primer*. W3C Recommendation 27 October 2009, 2009. Available from `http://www.w3.org/TR/owl2-primer/`.

[3] P. Hitzler, M. Krötzsch, and S. Rudolph. *Foundations of Semantic Web Technologies*. Chapman & Hall/CRC, 2009.

[4] T. A. Sudkamp. *Languages and Machines*. Addison Wesley, 3rd edition, 2006.