COT5310: Formal Languages and Automata Theory

Lecture Notes #3: Complexity

Dr. Ferucio Laurenţiu Ţiplea
Visiting Professor
School of Computer Science
University of Central Florida
Orlando, FL 32816
E-mail: tiplea@cs.ucf.edu
http://www.cs.ucf.edu/~tiplea

Complexity

1. Time and space bounded computations

2. Central complexity classes

3. Reductions and completeness

4. Hierarchies of complexity classes

1. Time and space bounded computations

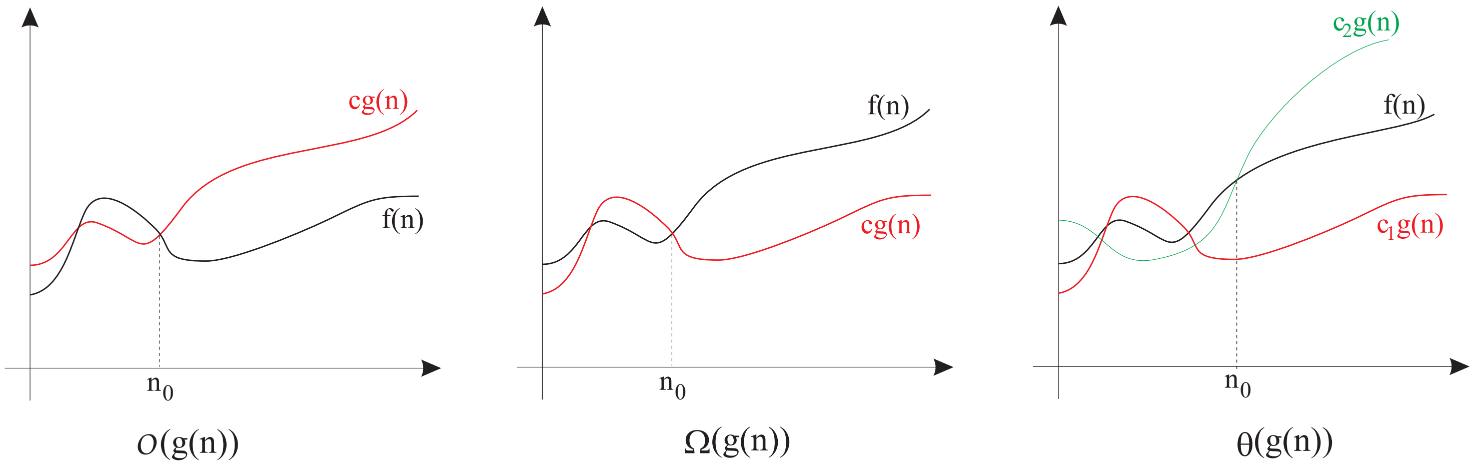
Let $g : \mathbf{N} \rightarrow \mathbf{R}_+$ be a function. Define the following sets:

$$\begin{aligned}
\mathcal{O}(g) &= \{f : \mathbf{N} \rightarrow \mathbf{R}_+ | (\exists c \in \mathbf{R}_+^*)(\exists n_0 \in \mathbf{N})(\forall n \geq n_0)(f(n) \leq cg(n))\} \\
\Omega(g) &= \{f : \mathbf{N} \rightarrow \mathbf{R}_+ | (\exists c \in \mathbf{R}_+^*)(\exists n_0 \in \mathbf{N})(\forall n \geq n_0)(cg(n) \leq f(n))\} \\
\Theta(g) &= \{f : \mathbf{N} \rightarrow \mathbf{R}_+ | (\exists c_1, c_2 \in \mathbf{R}_+^*)(\exists n_0 \in \mathbf{N})(\forall n \geq n_0) \\
&\qquad (c_1 g(n) \leq f(n) \leq c_2 g(n))\} \\
o(g) &= \{f : \mathbf{N} \rightarrow \mathbf{R}_+ | (\forall c \in \mathbf{R}_+^*)(\exists n_0 \in \mathbf{N})(\forall n \geq n_0)(f(n) \leq cg(n))\}
\end{aligned}$$

Let $f, g : \mathbf{N} \rightarrow \mathbf{R}_+$ and $X \in \{\mathcal{O}, \Omega, \Theta, o\}$. f is said to be X of g, denoted $f(n) = X(g(n))$, if $f \in X(g)$.

$\mathcal{O}$ ("big O"), Ω ("big Ω"), Θ ("big Θ"), and o ("little o") are order of magnitude symbols.

1.1. Orders of magnitude



- $f(n) = \mathcal{O}(g(n))$
  - $g(n)$ is an asymptotic upper bound for $f(n)$
  - $f(n)$ is no more than $g(n)$
  - used to state the complexity of a worst case analysis;
- $f(n) = \Omega(g(n))$ – similar interpretation;
- $f(n) = o(g(n))$ – $f(n)$ is less than $g(n)$ (the difference between $\mathcal{O}$ and o is analogous to the difference between $\leq$ and $<$).

Proposition 1 Let $f, g, h, k : \mathbf{N} \rightarrow \mathbf{R}_+$. Then:

(1) $f(n) = \mathcal{O}(f(n))$;

(2) if $f(n) = \mathcal{O}(g(n))$ and $g(n) = \mathcal{O}(h(n))$, then $f(n) = \mathcal{O}(h(n))$;

(3) $f(n) = \mathcal{O}(g(n))$ iff $g(n) = \Omega(f(n))$;

(4) $f(n) = \Theta(g(n))$ iff $f(n) = \mathcal{O}(g(n))$ and $f(n) = \Omega(g(n))$;

(5) if $f(n) = \mathcal{O}(h(n))$ and $g(n) = \mathcal{O}(k(n))$, then $(f \cdot g)(n) = \mathcal{O}(h(n)k(n))$ and $(f + g)(n) = \mathcal{O}(max\{h(n), k(n)\})$;

(6) if there exists $n_0 \in \mathbf{N}$ such that $g(n) \neq 0$ for any $n \geq n_0$, then $f(n) = o(g(n))$ iff $lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$.

1.1. Orders of magnitude

Some useful inequalities:

- (Stirling's formula)

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\frac{1}{12n+1}} \le n! \le \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\frac{1}{12n}},$$

  for any $n \ge 1$;

- for any real constants ϵ and c such that $0 < \epsilon < 1 < c$,

  $1 < ln\, ln\, n < ln\, n < e^{\sqrt{(ln\, n)(ln\, ln\, n)}} < n^\epsilon < n^c < n^{ln\, n} < c^n < n^n < c^{c^n}$

  (each inequality holds for all $n \ge n_0$, where n_0 is suitable chosen).

Example 1

1. if $f(x) = a_0 + a_1x + \cdots + a_kx^k$ is a polynomial of degree k with real coefficients and $f(x) \geq 0$ for any $x \in \mathbf{N}$, then $f(n) = \Theta(n^k)$;
2. $log_c\, n = \Theta(log\, n)$, for any real constant $c > 1$;
3. $log\, n = \mathcal{O}(n^\epsilon)$, for any real number ϵ such that $0 < \epsilon < 1$;
4. $log^k\, n = \mathcal{O}(n)$, for any natural number $k \geq 1$;
5. $n! = \Omega(2^n)$ and $n! = o(n^n)$;
6. $log(n!) = \Theta(nlog\, n)$.

Example 2 If $f : \mathbf{N} \rightarrow \mathbf{R}_+$ satisfies $f(n) \geq 1$ for any $n \geq n_0$ and some $n_0 \in \mathbf{N}$, then:

1. $\frac{1}{2} 2^{\lceil log_2 f(n) \rceil} \leq f(n) \leq 2^{\lceil log_2 f(n) \rceil}$, for any $n \geq n_0$;
2. $f(n) = \Theta(2^{\lceil log_2 f(n) \rceil})$.

Remark 1 $4^n \neq \mathcal{O}(2^n)$.

1.1. Orders of magnitude

Let $\mathcal{A}$ and $\mathcal{B}$ be sets of functions as those defined above (e.g., $\mathcal{O}(g)$ etc.), and let $f : \mathbf{N} \rightarrow \mathbf{R}_+$. Then, we denote

1. $f + \mathcal{A} = \{f + g | g \in \mathcal{A}\}$;
2. $\mathcal{A} + \mathcal{B} = \{f + g | f \in \mathcal{A},\ g \in \mathcal{B}\}$;
3. $f\mathcal{A} = \{f \cdot g | g \in \mathcal{A}\}$. If f is the constant c function, then we will write $c\mathcal{A}$ instead of $f\mathcal{A}$;
4. $\mathcal{A}\mathcal{B} = \{fg | f \in \mathcal{A},\ g \in \mathcal{B}\}$;
5. $\mathcal{O}(\mathcal{A}) = \bigcup_{f \in \mathcal{A}} \mathcal{O}(f)$.

Convention: $\mathcal{A} = \mathcal{B}$ stands for $\mathcal{A} \subseteq \mathcal{B}$.

Proposition 2 Let $f, g : \mathbf{R}_+ \mathbf{R}_+$ and $c \in \mathbf{R}_+$. Then:

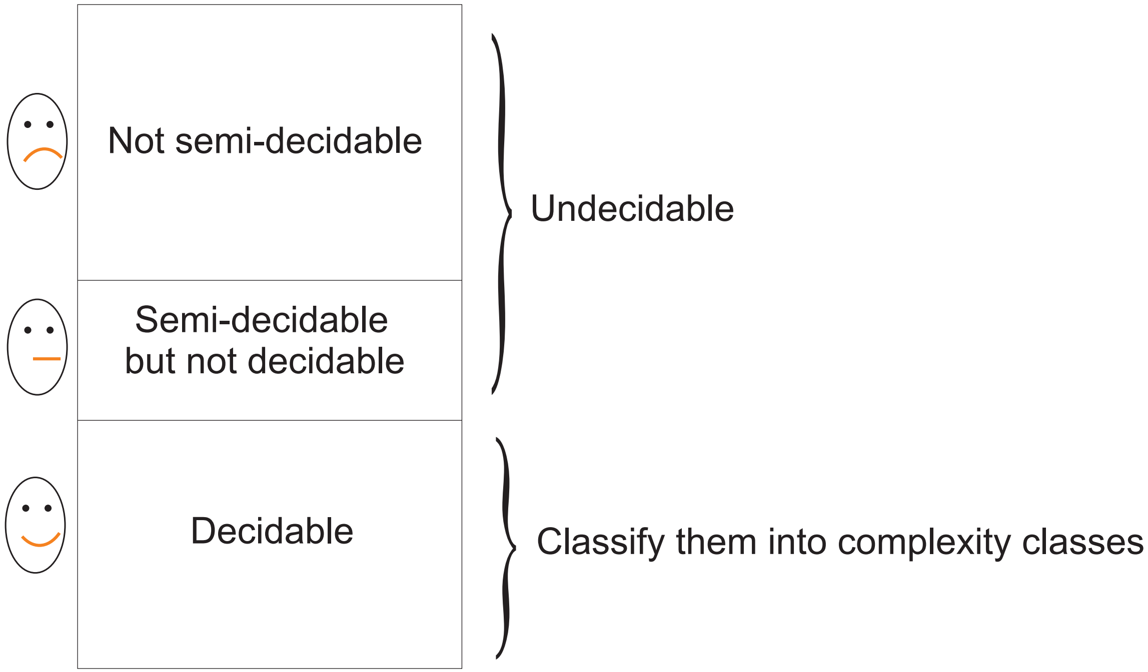
(1) $\mathcal{O}(f(n)) + \mathcal{O}(g(n)) = \mathcal{O}(f(n) + g(n))$;

(2) $c\mathcal{O}(f(n)) = \mathcal{O}(f(n))$;

(3) $\mathcal{O}(\mathcal{O}(f(n))) = \mathcal{O}(f(n))$;

(4) $\mathcal{O}(f(n))\mathcal{O}(g(n)) = \mathcal{O}(f(n)g(n))$;

(5) $\mathcal{O}(f(n)g(n)) = f(n)\mathcal{O}(g(n))$.



Decidable problems are classified into complexity classes according to the amount of work (time, space etc.) needed to solve them.

1.2. Running time and work space of TMs

Let M be a deterministic TM (DTM) that halts on all inputs, and let w be an input.

1. The computation time of M on w is the number of steps required by M to halt on w (regardless of whether M accepts or not w);

2. The running time or time complexity of M is the function $time_M$ given by:

   - $time_M(n)$ is the maximum computation time of M on inputs of length n, for any $n \geq 0$.

Assumption: $time_M(n) \geq n$, for any n (M needs at least n steps to read the entire input).

1.2. Running time and work space of TMs

Let M be a non-deterministic TM (NTM) wherein all computations halt on all words, and let w be an input.

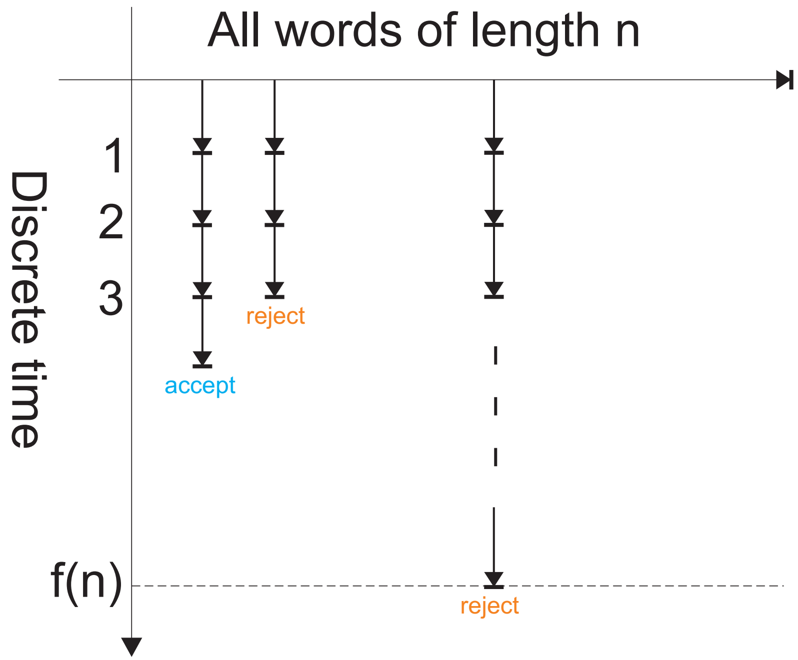
1. The computation time of M on w is the maximum number of steps required by M to halt on w (regardless of whether or not M accepts w);

2. The running time ot time complexity of M is the function $time_M$ given by:

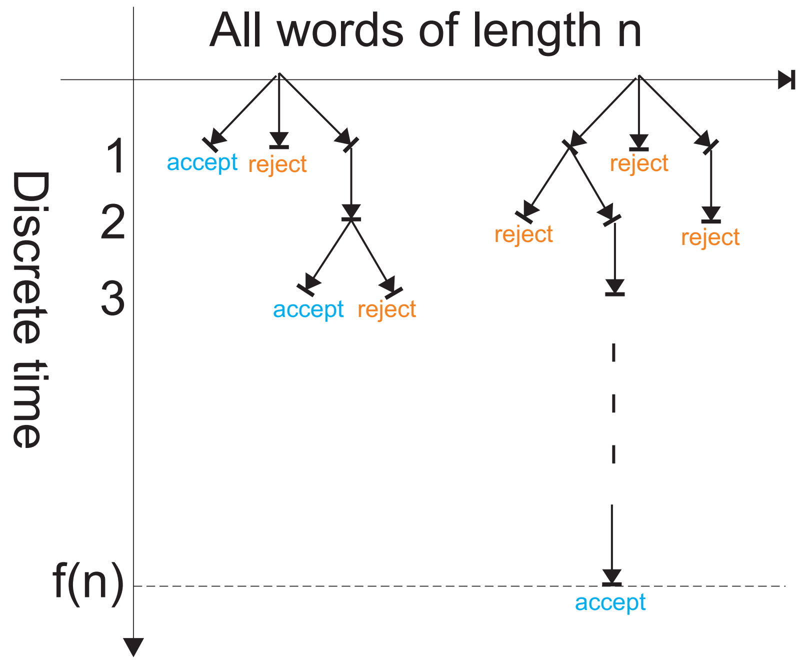
   - $time_M(n)$ is the maximum computation time of M on inputs of length n, for any $n \geq 0$.

Assumption: $time_M(n) \geq n$, for any n.

1.2. Running time and work space of TMs



DTM: computation threads



NTM: computation trees

Measuring deterministic and non-deterministic running time.

1.2. Running time and work space of TMs

Let $f : \mathbf{N} \rightarrow \mathbf{N}$. We say that a language L is accepted by a DTM (NTM) M within time $f(n)$ if M accepts L and the running time of M is at most $f(n)$ for every n.

Remark 2 The above concepts can be generalized to multiple-tape TM (mTM) as well.

1.2. Running time and work space of TMs

Theorem 1 (Reduction in the number of tapes)
If L is accepted by an mDTN within time $f(n)$ and $f(n) \geq n$ for any n, then L is accepted by a DTM within time $\mathcal{O}(f^2(n))$.

Proof Given an mDTN M, construct a DTM M' as follows. M' stores the contents of M's tapes on its single tape and uses a new symbol # to separate them. In addition, M' keeps track of the locations of M's heads by marking the corresponding cells on its tape.

To simulate a single move of M, M' scans its tape from left to right, records the marked symbols, and then makes a pass from right to left in order to update the tapes according to M's transition relation.

The total time to simulate one step of M is $\mathcal{O}(f(n))$. The initial stage (where M' puts its tape into the proper format) requires $\mathcal{O}(n)$ steps. Afterward, M' simulates each of the steps of M using $\mathcal{O}(f(n)))$ steps. Therefore, the entire simulation uses $\mathcal{O}(n) + \mathcal{O}(f^2(n))$ steps. As $f(n) \geq n + 1$, we conclude that M''s running time is $\mathcal{O}(f^2(n))$. $\square$

1.2. Running time and work space of TMs

Theorem 2 If L is accepted by an NTN within time $f(n)$ and $f(n) \geq n$ for any n, then L is accepted by a DTM within time $2^{\mathcal{O}(f(n))}$.

Proof Given an NTM M, construct a 3-tape DTM M' which tries all possible branches of M's non-deterministic computation. If M' ever finds an accept state on one of these branches, then accepts; otherwise, rejects.

On an input of length n, any computation tree of M has at most $c^{f(n)}$ leaves, for some constant c. The time for starting from the root and traveling down to a leaf node is $\mathcal{O}(f(n))$. Therefore, the running time of M' is

$$\mathcal{O}(f(n)c^{f(n)}) = 2^{\mathcal{O}(f(n))}.$$

By Theorem 1, M' can be converted into a DTM whose running time is

$$(2^{\mathcal{O}(f(n))})^2 = 2^{\mathcal{O}(2f(n))} = 2^{\mathcal{O}(f(n))}.$$

(note that M' does not need to count the numbers of moves of M because every computation of M on words of length n halts in $\mathcal{O}(f(n))$ steps). $\square$

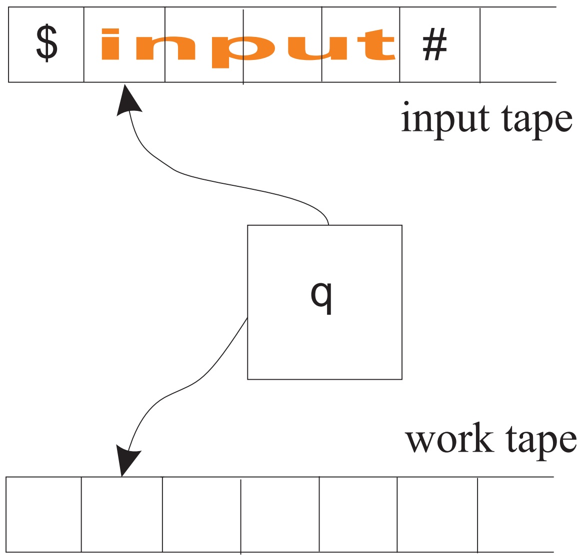
Remark 3 Some Turing machines may require only a very limited amount of work space, although their inputs are arbitrarily large. Therefore, to be able to talk about space bounds smaller than linear we have to count only the tape cells scanned by Turing machines during their computations, except for those used by inputs. To do this correctly, we need a suitable computation model.

An off-line Turing machine is a 2-tape TM with the following particularities:

- one of its tapes is read-only. It holds the input surrounded by two special symbols needed to detect the left-hand and the right-hand ends of the input;

- the other tape is a read/write work tape. It may be read and written in the usual way. Only the cells scanned on this tape contribute to the space complexity of the machine.

Off-line TMs may be deterministic or non-deterministic.



1.2. Running time and work space of TMs

Let M be an off-line DTM that halts on all inputs and let w be an input.

1. The computation space of M on w is the maximum number of tape cells scanned by M on the work tape during its computation on w;

2. The work space of M is the function $space_M$ given by:
   - $space_M(n)$ is the maximum computation space of M on inputs of length n, for any $n \geq 0$.

Assumption: $space_M(n) \geq 1$, for any n (M needs at least one tape cell to decide on its input).

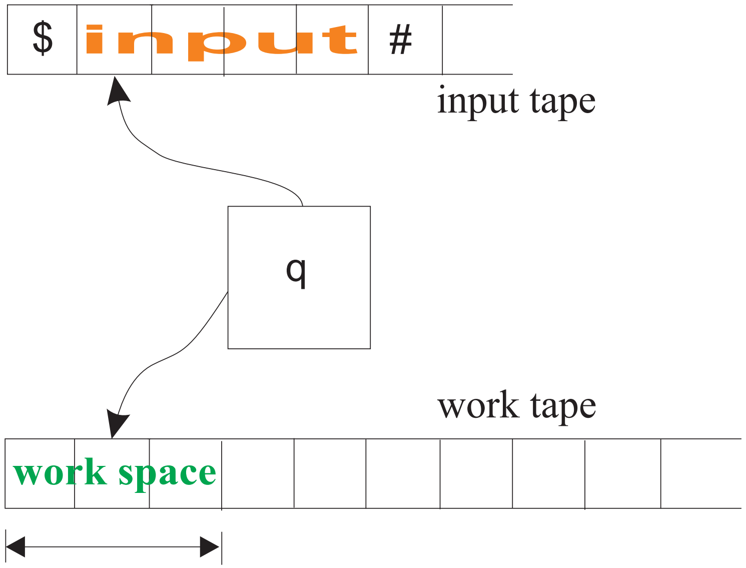
Let M be an off-line NTM wherein all computations halt on all inputs, and let w be an input.

1. The computation space of M on w is the maximum number of tape cells scanned by M on the work tape during any computation of M on w;

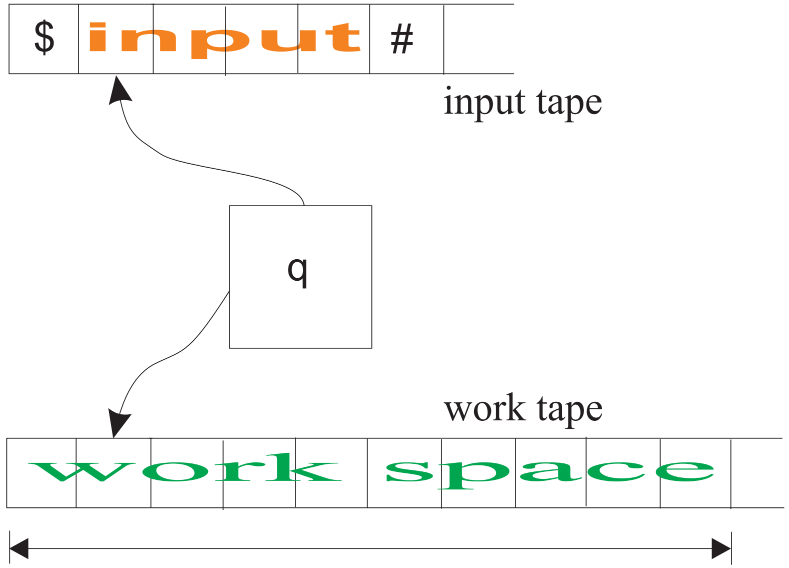
2. The work space of M is the function $space_M$ given by:
   - $space_M(n)$ is the maximum computation space of M on inputs of length n, for any $n \geq 0$.

Assumption: $space_M(n) \geq 1$, for any n.

1.2. Running time and work space of TMs



work space < input



work space > input

Let $f : \mathbf{N} \to \mathbf{N}$. We say that a language L is accepted by an off-line DTM (NTM) M within space $f(n)$ if M accepts L and the work space of M is at most $f(n)$ for every n.

Remark 4 The above concepts can be generalized to off-line TMs with m work tapes (off-line mTM).

Theorem 3 (Reduction in the number of tapes)
If L is accepted by an off-line mTM within space $f(n)$, then L is accepted by an off-line TM within space $f(n)$.

Remark 5 In order to study space bounds $f(n)$ satisfying $f(n) \geq n$ for all n, we may consider single-tape TMs. For such machines, space complexity is defined by counting the tape cells scanned by the machine on its single tape, including those used by the input.

2. Central complexity classes

2.1. Definitions and basic properties

2.2. Closure under complementation

2.3. P and NP

2.4. L and NL

2.5. $PSPACE$ and $NPSPACE$

2.6. $EXPTIME$ and $NEXPTIME$

2.7. Beyond $NEXPTIME$

2.1. Definitions and basic properties

A proper complexity function is any function f from $\mathbf{N}$ into $\mathbf{N}$ which satisfies:

- f is increasing ($f(n+1) \geq f(n)$ for any n);
- there exists an input-output TM that on inputs of length n writes $f(n)$ in unary on the output tape, and works within time $\mathcal{O}(n + f(n))$ and space $\mathcal{O}(f(n))$.

Example 3

1. $\log n$, $n \log n$, n^k, $\sqrt{n}$ are proper complexity functions;
2. Addition, multiplication, and exponentiation of proper complexity functions is a proper complexity function.

Remark 6 The class of proper complexity functions includes essentially all reasonable functions one would expect to use in the analysis of algorithms and the study of their complexity.

In what follows we will consider only proper complexity functions, although some results may hold for arbitrary functions from $\mathbf{N}$ into $\mathbf{N}$.

Time complexity classes

- $TIME(f(n))$ = the class of all languages that are decidable by DTMs within time $\mathcal{O}(f(n))$
  - $P = \bigcup_{k\geq 1} TIME(n^k)$
  - $EXPTIME = \bigcup_{k\geq 1} TIME(2^{n^k})$
- $NTIME(f(n))$ = the class of all languages that are decidable by NDTs within time $\mathcal{O}(f(n))$
  - $NP = \bigcup_{k\geq 1} NTIME(n^k)$
  - $NEXPTIME = \bigcup_{k\geq 1} NTIME(2^{n^k})$

Space complexity classes

- $SPACE(f(n))$ = the class of all languages that are decidable by DTMs within space $\mathcal{O}(f(n))$
  - $L = SPACE(\log n)$
  - $PSPACE = \bigcup_{k\geq 1} SPACE(n^k)$
- $NSPACE(f(n))$ = the class of all languages that are decidable by NDTs within space $\mathcal{O}(f(n))$
  - $NL = NSPACE(\log n)$
  - $NPSPACE = \bigcup_{k\geq 1} NSPACE(n^k)$

Directly from definitions it follows:

1. $TIME(f(n)) \subseteq NTIME(f(n))$;
2. $SPACE(f(n)) \subseteq NSPACE(f(n))$;

Proposition 3 $NTIME(f(n)) \subseteq SPACE(f(n))$.

Proof Let M be a NTM that works within time $f(n)$. Construct a DTM M' which simulates each computation tree of M in a DFS manner (it uses two extra tapes, one to simulate computations of M, and one to enumerate all choices for any configuration of M). As M' makes $\mathcal{O}(f(n))$ moves, M will scan $\mathcal{O}(f(n))$ tape cells on each work tape. $\square$

Proposition 4 If $f(n) \geq n$ for any n, then

$$NTIME(f(n)) \subseteq TIME(2^{\mathcal{O}(f(n))}).$$

Proof From Theorem 1. $\square$

Remark 7 $\mathcal{O}(2^{f(n)}) = 2^{\mathcal{O}(f(n))}$ but $2^{\mathcal{O}(f(n))} \neq \mathcal{O}(2^{f(n)})$. Therefore, Proposition 4 should be read as follows:

> for any $L \in NTIME(f(n))$ there exists a constant c such that $L \in TIME(2^{cf(n)})$.

Theorem 4 If $f(n) \geq \log n$ for any n, then

$$NSPACE(f(n)) \subseteq TIME(2^{\mathcal{O}(f(n))}).$$

Proof Let M be an off-line NTM working in space $g(n) = cf(n)$ for some constant c. A configuration of M is given by the current state, by the work tape, and by the positions of the two tape heads (the input is not part of any configuration!).

There are $|Q|(n+2)cf(n)|\Gamma|^{cf(n)}$ pairwise distinct configurations on words of length n. Moreover, by the hypothesis we get

$$|Q|(n+2)cf(n)|\Gamma|^{cf(n)} \leq 2^{df(n)} = 2^{\mathcal{O}(f(n))},$$

for some constant d.

Construct a DTM M' which decides whether some final configuration can be reach from the initial configuration in the configuration graph of M. The configuration graph has at most $2^{\mathcal{O}(f(n))}$ and this problem can be generously solved in quadratic time with respect to the number of nodes. $\square$

Theorem 5 (Savitch's Theorem)
If $f(n) \geq \log n$ for any n, then

$$NSPACE(f(n)) \subseteq SPACE(f^2(n)).$$

Conclusions:

$TIME(f(n)) \subseteq NTIME(f(n))$	$NTIME(f(n)) \subseteq TIME(2^{\mathcal{O}(f(n))})$	$f(n) \geq n$
$SPACE(f(n)) \subseteq NSPACE(f(n))$	$NSPACE(f(n)) \subseteq SPACE(f^2(n))$	$f(n) \geq \log n$
$TIME(f(n)) \subseteq SPACE(f(n))$	$NSPACE(f(n)) \subseteq TIME(2^{\mathcal{O}(f(n))})$	$f(n) \geq \log n$

Corollary 1

$L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE = NPSPACE \subseteq EXPTIME \subseteq NEXPTIME.$

Remark 8 We will prove later that at least one of the inclusions in Corollary 1 is proper (most researchers believe that all the inclusions are proper).

2.2. Closure under complementation

The complement of a decision problem is the decision problem resulting from reversing the yes and no answers.

Given a complexity class C, the complement class of C, denoted $co-C$, is the set of complements of every problem in C. Notice that this is not the complement of the complexity class itself as a set of problems.

A class is said to be closed under complementation if the complement of any problem in the class is still in the class.

2.2. Closure under complementation

Theorem 6 Deterministic complexity classes ($SPACE(f(n))$, $TIME(f(n))$) are closed under complementation.

Proof Add a last step reversing the answer. $\square$

The argument in the proof of Theorem 6 cannot be applied directly to nondeterministic complexity classes because if there exist both computation paths which accept and paths which reject, and all the paths reverse their answer, there will still be paths which accept and paths which reject. Consequently, the machine accepts in both cases.

One of the most surprising complexity results shown to date is that $NSPACE$ is closed under complementation.

2.2. Closure under complementation

Theorem 7 (Immerman-Szelepcsényi)
$NSPACE(f(n)) = co - NSPACE(f(n)$, for any $f(n) \geq \log n$.

Proof Claim 1: Given a graph G and a node x, the number of nodes reachable from x can be computed by an NTM within space $\log n$.

Claim 2. Let M be an off-line NTM that works within space $\mathcal{O}(f(n))$. Then, there exists an off-line NTM M' such that, for any input of length n, M' can decide within space $\mathcal{O}(f(n))$ if M rejects the input. □

This theorem was proved independently by Neil Immerman (the general case) and Robert Szelepcsényi (for the particular case of context-sensitive languages - we will see later that $\mathcal{L}_1 = NSPACE(n)$).

2.2. Closure under complementation

Neil Immerman tells a story about Robert Szelepcsényi:

> Szelepcsényi's result that Context Sensitive Languages are closed under complementation was announced in an issue of EATCS (in the same issue, two other articles mentioned Immerman's own proof that NSPACE is closed under complementation, an effectively equivalent result). Szelepcsnyi was an undergrad at the time, and his adviser gave him the famous problem as a challenge, probably not really expecting him to actually solve it. He did solve it, perhaps because he was never told that it was an old open problem that others had failed to solve it.

From: "Computational Complexity" at http://weblog.fortnow.com/

2.3. P and NP

$$\boxed{P = \cup_{k\geq 1} TIME(n^k)}$$

The class P plays a central role in the complexity theory because:

- P is invariant for all models of computations that are polynomially equivalent to the single-tape DTMs;
- P "corresponds" to the class of problems that are realistically solvable on a computer.

We will give examples of problems in P (NP etc.) by using a high-level description of the algorithms which avoids tedious details of tapes and head motions. To do that we need to follow certain conventions:

- algorithms perform in stages;
- a stage is analogous to a step of a Turing machine (though of course, implementing a stage of an algorithm on a TM will require many TM steps);

Graph Reachability Problem (GRP)

Instance: A graph $G = (V, E)$ and two nodes x and y;

Question: Is there a path from x to y?

Given a graph $G = (V, E)$ and two nodes x and y, define the following sequence of sets:

- $V_0 = \{x\}$;
- $V_{i+1} = V_i^\bullet$, for any $i \geq 0$.

Properties:

1. there exists i and j such that $j < i$ and $V_i = V_j$;
2. y is reachable from x iff there exists k such that $y \in V_k$.

<u>Algorithm $\mathcal{A}_1$</u>

input: graph $G = (V, E)$ and $x, y \in V$;

output: "yes", if y is reachable from x, and "no", otherwise;

```
begin
  mark x;
  S := {x};
  repeat
    choose a ∈ S;
    S := S − {a};
    mark all unmarked immediate successors of a and add
            them to S;
  until y is get marked or there is no unmarked immediate
            successor of a;
  if y is marked then "yes" else "no";
end.
```

How is the element a chosen among all elements in S? The choice affects the style of search:

- if S is implemented as a queue, then the resulting search is breadth-first (BFS);
- if S is implemented as a stack, then the resulting search is depth-first (DFS).

Complexity:

- time complexity: $GRP \in TIME(n^2)$;
- space complexity: $GRP \in SPACE(n)$.

As a conclusion, $GRP \in P$.

$$\boxed{NP = \cup_{k \geq 1} NTIME(n^k)}$$

NP is an important complexity class because it contains many problems of practical interest.

All the problems in this class rely on brute-force search techniques that can be completed non-deterministically in polynomial time (attempts to avoid brute-force search in these problems have not been successful so far).

Boolean formulas over a finite set X of boolean variables are defined by:

- x is a boolean formula, for every $x \in X$;
- if φ_1 and φ_2 are boolean formulas, then $\neg\varphi_1$, $(\varphi_1 \vee \varphi_2)$, and $(\varphi_1 \wedge \varphi_2)$ are boolean formulas.

A truth assignment for X is any function $\gamma : X \rightarrow \{0, 1\}$. A truth assignment γ satisfies a boolean formula φ, denoted $\gamma \vdash \varphi$, if φ evaluates to the truth value true (1) when each variable x in φ is replaced by $\gamma(x)$.

A boolean formula φ is valid if it is satisfied by all assignments. It is clear that φ is unsatisfiable iff $\neg\varphi$ is valid.

A boolean formula φ is in conjunctive normal form (CNF) if

$$\varphi = c_1 \wedge \cdots \wedge c_k,$$

where, for any i, c_i is of the form

$$c_i = l_1^i \vee \cdots \vee l_{m_i}^i$$

and each l_j^i is either a variable or the negation of some variable. c_i are called clauses and l_j^i are called literals. If $l_j^i = x \in X$, then it is a positive literal; otherwise, it is a negative literal. If $m_i \leq s$ for any i, then φ is called in s-conjunctive normal form (s-CNF).

Theorem 8 Every boolean formula is equivalent to one in CNF.

Satisfiability Problem (SAT)

Instance: A boolean formula φ over a set $X = \{x_1, \ldots, x_n\}$ of boolean variables;

Question: Is there a truth assignment for X that satisfies φ?

Algorithm $\mathcal{A}_2$

input: boolean formula φ over $X = \{x_1, \ldots, x_n\}$;
output: "yes" if φ is satisfiable, and "no", otherwise;
begin
  choose non-deterministically a truth assignment for X;
  if φ evaluates to the truth value true under the assignment
    then "yes" **else** "no";
end.

Therefore, $SAT \in NP$.

$$\boxed{L = SPACE(\log n)}$$

A palindrome over an alphabet Σ is any word $w \in \Sigma^*$ such that $w = x\tilde{x}$, for some word x ($\tilde{x}$ is the mirror image of x).

Palindrome Problem (PAL)

Instance: An alphabet Σ and a word $w \in \Sigma^*$;

Question: Is w a palindrome?

Theorem 9 $PAL \in L$.

Proof Construct an off-line DTM with 2 work tapes which counts in binary and checks the i-th input symbol from left to right against the i-th input symbol from right to left. If input has length n, the machine scans $\mathcal{O}(\log n)$ cells on each work tape. $\square$

$$\boxed{NL = NSPACE(\log n)}$$

Theorem 10 $GRP \in NL$.

Proof Given a GRP instance $(G = (V, E), x, y)$, construct an off-line DTM with 2 work tapes as follows:

- nodes are written in binary;
- write x on the first work tape;
- choose non-deterministically a node z, write it on the second work tape, and check whether $(x, z) \in E$. If $(x, z) \notin E$ then halt and reject. If $z = y$ then halt and accept; otherwise, replace x by z and repeat the procedure, unless the machine has gone on for n steps and rejects, where n is the number of nodes.

$\square$

2-Satisfiability Problem (2SAT)

Instance: A 2-CNF boolean formula φ over a set $X = \{x_1, \ldots, x_n\}$ of boolean variables;

Question: Is there a truth assignment for X that satisfies φ?

Given an instance φ of $2SAT$, define a graph $G(\varphi)$ as follows:

- the nodes of G are the variables of φ and their negations;
- there is an arc (α, β) iff there is a clause $(\neg\alpha \vee \beta)$ (or $(\beta \vee \neg\alpha)$) (intuitively, these edges capture the logical implications).

$G(\varphi)$ has an interesting property: if (α, β) is an edge, then so is $(\neg\beta, \neg\alpha)$.

Theorem 11 Let φ be an instance of $2SAT$. φ is unsatisfiable iff there exists a variable x such that there are paths from x to $\neg x$ and from $\neg x$ to x.

Corollary 2 $2SAT$ is in NL.

$$\boxed{NPSPACE = PSPACE = \bigcup_{k\geq 1} SPACE(n^k)}$$

Quantified boolean formulas (QBF) are defined as boolean formulas but with the difference that "$\forall x$" and "$\exists x$" may precede any (sub-)formula. For instance,

$$(\forall x)(\exists y)((x \vee y) \wedge (\forall z)(x \vee z))$$

is a QBF.

In the formula $(Qx)\varphi$, where $Q \in \{\forall, \exists\}$, φ is called the scope of the quantifier Q. A QBF is closed or fully quantified if each variable appears within the scope of some quantifier.

If all quantifiers of a formula appear at the beginning of the formula, then the formula is called in prenex normal form (PNF). Any QBF may be put into an equivalent PNF.

Quantified Satisfiability Problem (QSAT)

Instance: A closed QBF φ in PNF;
Question: Is φ true?

(the problem is also known as QBF; we use QSAT to emphasize that it is yet another version of SAT).

The main difference between SAT and QSAT:

- SAT asks to decide if there exists a truth value;
- QSAT asks to decide if there exists a set of truth value. For example, the QBF $\varphi = (\forall x)(\exists y)((x \vee y) \wedge (x \vee \neg y))$ is true if there exists two truth assignment, one of them having 0 substituted for x and the other one having 1 substituted for x.

SAT can be viewed as a particular case of QSAT:

- to each SAT instance φ associate the QBF instance $(\exists x_1)\cdots(\exists x_n)\varphi$.

Theorem 12 $QSAT \in PSPACE$.

Proof Consider the algorithm $\mathcal{A}$ which, on input φ, performs as follows:

1. if φ does not contain quantifiers (i.e., it is an expression with only constants), evaluate it and accept (reject) if it is true (false);
2. if $\varphi = (\exists x)\psi$ then recursively call $\mathcal{A}$ on ψ, fisrt with 0 substituted for x, and then with 1 substituted for x. If either result is accept, then accept; otherwise, reject;

3. if $\varphi = (\forall x)\psi$ then recursively call $\mathcal{A}$ on ψ, first with 0 substituted for x, and then with 1 substituted for x. If both results are accept, then accept; otherwise, reject.

The depth of the recursion is at most the number of variables. At each level we need only store the value of one variable. Therefore, the total space used is $\mathcal{O}(n)$, where n is the number of variable. Therefore, $\mathcal{A}$ runs in linear space. $\square$

A game is a competition in which opposing parties attempts to achieve some goal according to some given rules.

Formula game: closed QBF in PNF

$$(\exists x_1)(\forall x_2)(\exists x_3)\cdots(Qx_n)\varphi,$$

where $Q = \exists$ if n is odd, and $Q = \forall$, otherwise.

Any closed QBF in PNF can be viewed as a formula game (to ensure strict alternation we may insert to the prefix appropriately quantified "dummy" variables that do not appear in φ).

Let

$$(\exists x_1)(\forall x_2)(\exists x_3) \cdots (Qx_n)\varphi,$$

be a formula game. We associate a game to it as follows:

1. two players A and B take turns selecting values of the variables $x_1, \ldots, x_n$;
2. player A (B) selects values for the variables that are bound to $\forall$ ($\exists$) quantifiers;
3. the order of play is the same as that of the quantifiers at the beginning of the formula;
4. if the formula is evaluated to the truth value true, then B wins; otherwise, A wins.

Example 4 Let

$$\varphi = (\exists x_1)(\forall x_2)(\exists x_3)((x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3))$$

- If B picks $x_1 = 1$, A picks $x_2 = 0$, and B picks $x_3 = 1$, then B wins;
- B can always win if he/she selects $x_1 = 1$ and $x_3 = \neg x_2$, where x_2 is A's choice. We say in this case that B has a winning strategy.

If we replace the third clause of φ by $(x_2 \vee \neg x_3)$, then A has a winning strategy ($x_2 = 0$ A's choice makes the formula false, no matter what B selects).

Formula Game Problem (GAME)

Instance: A formula game φ;

Question: Does player B have a winning strategy in the game associated with φ?

Corollary 3 $GAME \in PSPACE$.

See http://www.ics.uci.edu/~eppstein/cgt/ for details on computational complexity of games such as GO, chess etc.

Let $G = (V, T, S, P)$ be a grammar.

1. G is context-sensitive if each rule is of the form:
   - $\alpha A\gamma \rightarrow \alpha\beta\gamma$, where $A \in V$, $\alpha, \gamma \in (V \cup T)^*$, and $\beta \in (V \cup T)^+$, or
   - $S \rightarrow \lambda$, and if this rule occurs then S does not appear on the right hand side of any rule in P.
2. G is length-increasing or monotonic if each rule is of the form:
   - $\alpha \rightarrow \beta$ with $\alpha \in (V \cup T)^* V (V \cup T)^*$, $\beta \in (V \cup T)^+$, and $|\alpha| \leq |\beta|$, or
   - $S \rightarrow \lambda$, and if this rule occurs then S does not appear on the right hand side of any rule in P.

It is known that context-sensitive grammars and monotonic grammars are equivalent.

A linear bounded automaton (LBA) is a non-deterministic Turing machine which works within space $\mathcal{O}(n)$.

Theorem 13 A language L is context-sensitive iff there exists an LBA which accepts L.

If $\mathcal{L}_1$ denotes the class of context-sensitive languages, then $\mathcal{L}_1 = NSPACE(n)$.

$$\boxed{EXPTIME = \bigcup_{k\geq 1} TIME(2^{n^k})}$$

Unary logic programs:

- Let Σ be a set consisting of one constant symbol $\perp$ and finitely many unary function symbols;
- Let $Pred$ be a finite set of unary predicate symbols, and x be a variable;
- A unary logic program over Σ, $Pred$, and x is a finite set of clauses of the form

$$p_0(t_0) \leftarrow p_1(t_1), \ldots, p_n(t_n)$$

or

$$p_0(t_0) \leftarrow true,$$

where $p_0, \ldots, p_n \in Pred$, and $t_0, \ldots, t_n$ are terms over $\Sigma \cup \{x\}$ with t_0 being flat, that is, $t_0 \in \{\perp, x, f(x) | f \in \Sigma - \{\perp\}\}$. Moreover, all clauses with $p_0(\perp)$ in the head have only $true$ in the body.

An atom is a construct of the form $p(t)$, where $p \in Pred$ and t is a term. If t is a ground term, that is, it does not contain x, then $p(t)$ is called a ground atom.

A proof tree for a ground atom $p(t)$ under a unary logic program LP is any tree that satisfies:

- its nodes are labeled by ground atoms;
- the root is labeled by $p(t)$;
- each intermediate node which is labeled by some B has children labeled by $B_1, \ldots, B_n$, where $B \leftarrow B_1, \ldots, B_n$ is a ground instance of a clause in LP (i.e., the variable x is substituted by ground terms over Σ);
- all the leaves are labeled by $true$.

Membership Problem for Unary Logic Programs (ULP)

Instance: A logic program LP and a ground atom $p(t)$;

Question: Is there a proof tree for $p(t)$ under LP?

Theorem 14 $ULP \in EXPTIME$.

Let $\mathcal{P}$ be a protocol, $T \subseteq \mathcal{T}_0$ a finite set, and $k \geq 1$.

- A run of $\mathcal{P}$ is called a (T,k)-run if all terms in the run are built up upon T and all messages communicated in the course of the run have length at most k.
- When for $\mathcal{P}$ only (T,k)-runs are considered we will say that it is a protocol under (T,k)-runs or a (T,k)-bounded protocol, and denote this by $(\mathcal{P},T,k)$.
- The (initial) secrecy problem for such protocols is formulated with respect to (T,k)-runs only, by taking into consideration the set T instead of $\mathcal{T}_0$.

Let $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w)$ be a (T, k)-bounded protocol. Then:

1. the number of messages communicated in the course of any (T, k)-run is bounded by
$$k^3|T|^{\frac{k+1}{2}} = 2^{3\log k + \frac{k+1}{2}\log|T|};$$
2. the number of instantiations (substitutions) of a given role u of $\mathcal{P}$ with messages of length at most k over T is bounded by
$$(2^{3\log k + \frac{k+1}{2}\log|T|})^{|u|(\frac{k+1}{2}+2)}$$
(u has exactly $|u|$ actions, and each action has at most $\frac{k+1}{2} + 2$ elements that can be substituted);

3. the number of (T,k)-*events* (i.e., events that can occur in all (T,k)-runs) is bounded by

   number of (T,k)-events $\leq$

$$\begin{array}{rl} \leq & \sum_{u\in role(\mathcal{P})} |u| \cdot 2^{(3\log k+\frac{k+1}{2}\log|T|)|u|(\frac{k+1}{2}+2)} \\ \leq & \sum_{u\in role(\mathcal{P})} |u| \cdot 2^{(3\log k+\frac{k+1}{2}\log|T|)|w|(\frac{k+1}{2}+2)} \\ = & |w| \cdot 2^{(3\log k+\frac{k+1}{2}\log|T|)|w|(\frac{k+1}{2}+2)} \\ = & 2^{\log|w|+(3\log k+\frac{k+1}{2}\log|T|)|w|(\frac{k+1}{2}+2)} \end{array}$$

   where $role(\mathcal{P})$ is the set of all roles of $\mathcal{P}$.

Define the size of $\mathcal{P}$ by:

$$size(\mathcal{P}) = |w| + \frac{k+1}{2}\log|T|.$$

Algorithm A1

```
input:   bounded protocol (P, T, k) without freshness check;
output:  "leaky protocol" if P has some leaky (T, k)-run w.r.t. initial secrets,
         and "non-leaky protocol", otherwise;
begin
 let E' be the set of all (T, k)-events;
 ξ := λ;  s := s_0;
 repeat
   E := E';
   E' := ∅;
   bool := 0;
   while E ≠ ∅ do
    begin
     choose e ∈ E;
     E := E − {e};
     if (s, ξ)[e⟩(s', ξe) then
        begin
         s := s';  ξ := ξe;  bool := 1;
        end
      else E' := E' ∪ {e};
    end
 until bool = 0;
 if (⋃_{A∈Ho} Secret_A) ∩ analz(s_I) ≠ ∅ then "leaky protocol" else "non-leaky protocol"
end.
```

Theorem 15 The initial secrecy problem for bounded protocols without freshness check is in $EXPTIME$.

Proof The algorithm $A1$ performs in exponential time with respect to the size of the protocol. $\square$

$$\boxed{NEXPTIME = \bigcup_{k\geq 1} NTIME(2^{n^k})}$$

Theorem 16 If $P = NP$ then $EXPTIME = NEXPTIME$.

Proof Let $L \in NEXPTIME$ and M an NTM which accepts L and works within time 2^{n^k}. Define the language:

$$L' = \{xB^{2^{|x|^k}-|x|} | x \in L\},$$

where B is a new symbol (the blank symbol). Show that:

- $L' \in NP$;
- By the hypothesis ($P = NP$), $L' \in P$;
- $L \in EXPTIME$.

$\square$

A grammar $G = (V, T, S, P)$ is called growing if there exists a function $f : (V \cup T)^* \rightarrow \mathbf{N}$ such that $f(\alpha) < f(\beta)$, for any rule $\alpha \rightarrow \beta \in P$. f is called a weight function for G.

A weight function f of G is minimal if

$$\sum_{a \in V \cup T} f(a) \leq \sum_{a \in V \cup T} f'(a)$$

for any weight function f' of G.

Proposition 5 For every growing grammar G there exists a minimal weight function f such that $f(a) \leq 2^{poly(|V|+|T|)}$, for some polynomial $poly$ and any $a \in V \cup T$.

<u>Variable Membership Problem for Growing Grammars (VMGG)</u>

Instance: A growing grammar G and a string w;

Question: Is w derivable in G?

Theorem 17 $VMGG \in NEXPTIME$.

See also `VariableMembershipProblem.pdf`.

2.7. Beyond $NEXPTIME$

There is no reason to stop at $NEXPTIME$ ...

- $EXPSPACE = \bigcup_{k\geq 1} SPACE(2^{n^k})$
- $NEXPSPACE = \bigcup_{k\geq 1} NSPACE(2^{n^k})$
- $2-EXPTIME = \bigcup_{k\geq 1} TIME(2^{2^{n^k}})$
- $2-NEXPTIME = \bigcup_{k\geq 1} NTIME(2^{2^{n^k}})$
- $3-EXPTIME = \bigcup_{k\geq 1} TIME(2^{2^{2^{n^k}}})$
- $3-NEXPTIME = \bigcup_{k\geq 1} NTIME(2^{2^{2^{n^k}}})$
- and so on.

Languages in the class $\bigcup_{n\geq 0} 0-EXPTIME$ are called elementary.

3. Reduction and Completeness

Reducibility:

- tool for exploring the relationship between problems;
- formalizes the concept of the most difficult problem in a complexity class;
- creates a semi-lattice structure which is one of the most important sources of intuition about complexity classes.

Two basic reductions:

- Karp (polynomial time many-one);
- log-space.

A is polynomially time many-one reducible or Karp reducible to B, denoted $A \leq_m B$, if there exists a function $f : \Sigma^* \rightarrow \Sigma^*$ computable in deterministic polynomial time and such that $x \in A$ iff $f(x) \in B$, for any x.

Proposition 6

1. $\leq_m$ is a pre-order;
2. $A \leq_m B$ iff $\bar{A} \leq_m \bar{B}$;
3. P, NP, $PSPACE$, $EXPTIME$, and $NEXPTIME$ are closed under m-reducibility.

3. Reduction and Completeness

Given $A, B \subseteq \Sigma^*$, we say that A is s-space reducible to B, denoted $A \leq_s B$, if there exists a function $f : \Sigma^* \rightarrow \Sigma^*$ such that:

1. $f(x)$ is computable in space $s(|x|)$;
2. $x \in A$ iff $f(x) \in B$, for any $x \in \Sigma^*$;
3. there exists a positive integer c such that $s(|f(x)|) \leq cs(|x|)$, for any $x \in \Sigma^*$.

An important case is when s is the log function, which will be called the log-space reducibility.

Condition 3 above assures the transitivity of $\leq_s$ (it avoids a space bounded machine write an output substantially larger than allowed by its space bound). This condition holds for the case $s = log$.

Proposition 7

1. $\leq_s$ is a pre-order;
2. $SPACE(s)$ and $NSPACE(s)$ are closed under $\leq_s$;
3. All complexity classes beyond L, and including L too, are closed under log-space reducibility.

Proposition 8 $A \leq_{log} B$ implies $A \leq_m B$.

$\leq_{log}$ allows a more refined theory about the relationships between problems. For this reason, people use $\leq_{log}$ whenever they can (which is almost always).

3. Reduction and Completeness

Let $\mathcal{C}$ be a class of languages and A a language.

1. A is m-hard for $\mathcal{C}$ if $B \leq_m A$, for any $B \in \mathcal{C}$;
2. A is m-complete for $\mathcal{C}$ if A is m-hard for $\mathcal{C}$ and $A \in \mathcal{C}$.

log-space hardness and log-space completeness are defined analogously.

Proposition 9 Let $\mathcal{C}$ be a complexity class.

1. If A is m-hard (log-space hard) for $\mathcal{C}$ and $A \leq_m B$ ($A \leq_{log} B$) then B is m-hard (log-space hard) for $\mathcal{C}$.
2. If A is m-hard (log-space hard) for $\mathcal{C}$ then $\bar{A}$ is m-hard (log-space hard) for $co - \mathcal{C}$.
3. If A is m-complete (log-space complete) for $\mathcal{C}$, $B \in \mathcal{C}$, and $A \leq_m B$ ($A \leq_{log} B$) then B is m-complete (log-space complete) for $\mathcal{C}$.

Unless otherwise specified, we will use m-hardness and m-completeness for NP and classes beyond NP, and log-space hardness and log-space completeness for P, NL, and L. For this reason, we will simply say "A is complete for $\mathcal{C}$" or "A is $\mathcal{C}$-complete".

Corollary 4 Let $\mathcal{C}'$ and $\mathcal{C}$ complexity classes such that $\mathcal{C}' \subseteq \mathcal{C}$. If A is complete for $\mathcal{C}$ and $A \in \mathcal{C}'$, then $\mathcal{C}' = \mathcal{C}$.

Complete problems for L

Any problem in L is log-space complete for L. This results is completely uninteresting because a reduction is meaningful only within a class that is computationally stronger than the reduction.

To categorize the languages in L we need weaker definitions of reductions.

Complete problems for NL

Theorem 18 GRP is NL-complete.

Proof Let $A \in NL$ and M be an NTM which decides A within space $\log n$. The reachability graph of M on an input of length n can be constructed in space $\log n$. We can assume that this graph has a single accepting configuration (node). This graph, together with the initial and accepting configurations, forms an instance of GRP. Moreover, $x \in A$ iff the accepting configuration in reachable from the initial configuration in the reachability graph of M on x. $\square$

Theorem 19 $2SAT$ is NL-complete.

Proof $\overline{GRP} \in co-NL = NL$ implies that $\overline{GRP}$ is NL-complete.

Reduce then $\overline{GRP}$ to $2SAT$. $\square$

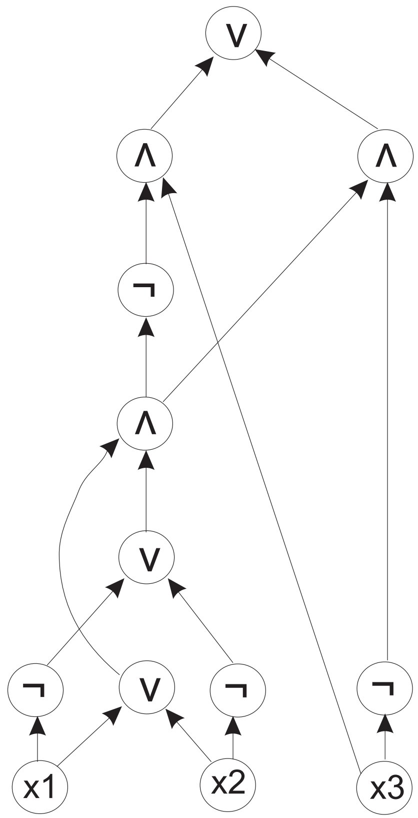
Complete problems for P

A boolean circuit is a graph $G = (V, E)$, where:

- $V = \{1, \ldots, n\}$, for some n;
- $(i, j) \in E$ implies $i < j$;
- there are no cycles;
- each node i in the graph, also called a gate, has associated a sort $s(i) \in \{true, false, \vee, \wedge, \neg\} \cup \{x_1, x_2, \ldots\}$ and has a number of input and output edges corresponding to its sort.

  Gates with no incoming (outgoing) edges are called input gates (output gates).

A boolean circuit is evaluated by assigning boolean values to variables and evaluating the gates from input to output nodes in a straightforward manner.



Given a boolean formula φ, there is a simple way to construct a boolean circuit G_φ such that, for any assignment γ, γ satisfies φ iff G_φ is evaluated to true under γ.

That is, SAT and $CIRCUIT\ SAT$ are equivalent.

The circuit value problem (CIRCUIT VALUE)

Instance: A boolean circuit G and a truth assignment γ;

Question: Compute the value of G under γ.

Theorem 20 $CIRCUIT\ VALUE$ is P-complete.

Complete problems for NP

Theorem 21 (Cook's Theorem)
SAT is NP-complete.

$CIRCUIT\ SAT$ is NP-complete too.

See SixBasicNP-completeProblems.pdf.

Complete problems for $PSPACE$

Theorem 22 $QSAT$ is $PSPACE$-complete.

The containment problem for FA (CP)

Instance:	Two finite automata A_1 and A_2;
Question:	Does $L(A_1) \subseteq L(A_2)$ hold?

Theorem 23 CP is $PSPACE$-complete.

Complete problems for $EXPTIME$

Theorem 24 ULP is $EXPTIME$-complete.

Theorem 25 The initial secrecy problem for bounded protocols without freshness check is $EXPTIME$-complete.

Proof Reduce ULP to the initial secrecy problem. $\square$

Space and time constructible functions:

- A function $f(n)$ is said to be time constructible if there exists an $f(n)$ time-bounded DTM that for each n has an input of length n on which it makes exactly $f(n)$ moves.
- A function $f(n)$ is said to be fully time constructible if there exists a DTM that makes exactly $f(n)$ moves on each input of length n.

In a similar way one can define space constructibility and fully space constructibility.

Space and time constructible functions:

- A function $f(n)$ is said to be time constructible if there exists a $f(n)$ time-bounded DTM that for each n has an input of length n on which it makes exactly $f(n)$ moves.
- A function $f(n)$ is said to be fully time constructible if there exists a DTM that makes exactly $f(n)$ moves on each input of length n.

In a similar way one can define space constructibility and fully space constructibility.

Theorem 26 (Hartmanis, Lewis, Sterns)
Let $f_1(n)$ and $f_2(n)$ be two functions such that:

- $f_1(n) \geq \log n$;
- $f_2(n)$ is fully space constructible;
- $\inf(f_1(n)/f_2(n)) = 0$.

Then, there exists a language L such that

$$L \in SPACE(f_2(n)) - SPACE(f_1(n)).$$

Corollary 5 Under the hypothesis of Theorem 26 and the supplementary assumption that $f_1(n) \leq f_2(n)$, we obtain

$$SPACE(f_1(n)) \subset SPACE(f_2(n)).$$

Corollary 6 For any $k \geq 1$,

$$SPACE(n^k) \subset SPACE(n^{k+1}).$$

Corollary 7 $NL \subset PSPACE$.

Proof Use Savitch's Theorem. □

Corollary 8 $PSPACE \subset EXPSPACE$.

Corollary 9 There are problems in $PSPACE$ requiring an arbitrarily large exponent to solve. Therefore, $PSPACE$ does not collapse to $SPACE(n^k)$, for some constant k.

Corollary 10 $L \neq SPACE(\log^2 n)$. Therefore, $L \neq NL$ or $NL \neq SPACE(\log^2 n)$.

Theorem 27 Let $f_1(n)$ and $f_2(n)$ be two functions such that:

- $f_2(n)$ is fully time constructible;
- $\inf(f_1(n) \log f_1(n)/f_2(n)) = 0$.

Then, there exists a language L such that

$$L \in TIME(f_2(n)) - TIME(f_1(n)).$$

Corollary 11 Under the hypothesis of Theorem 27 and the supplementary assumption that $f_1(n) \leq f_2(n)$, we obtain

$$TIME(f_1(n)) \subset TIME(f_2(n)).$$

Corollary 12 $TIME(2^n) \subset TIME(n^2 2^n)$.

Theorem 27 cannot be applied to

$$f_1(n) = 2^n \quad \text{and} \quad f_2(n) = n2^n$$

.

Lemma 1 (Translation lemma)
Let $f_1(n)$, $f_2(n)$, and $g(n)$ be three functions such that:

- f_1, f_2, and g are fully space constructible;
- $f_2(n) \geq n$ and $g(n) \geq n$.

For any $K \in \{SPACE, TIME, NSPACE, NTIME\}$, if

$$K(f_1(n)) \subseteq K(f_2(n)),$$

then

$$K(f_1(g(n))) \subseteq K(f_2(g(n))).$$

Corollary 13 $TIME(2^n) \subset TIME(n2^n)$.

Corollary 14 $NSPACE(n^r) \subset NSPACE(n^{r+\epsilon})$, for any real numbers $r \geq 1$ and $\epsilon > 0$.