Complexity Theory

Lecture 10

Tom Gur
http://www.cl.cam.ac.uk/teaching/2324/Complexity
A function $f$ is called a one way function if it satisfies the following conditions:

1. $f$ is one-to-one.

We cannot hope to prove the existence of one-way functions without at the same time proving $P \neq NP$.

It is strongly believed that the RSA function:

$$f(x, e, p, q) = (x^e \mod pq, pq, e)$$

is a one-way function.
A function $f$ is called a one way function if it satisfies the following conditions:

1. $f$ is one-to-one.
2. for each $x$, $|x|^{1/k} \leq |f(x)| \leq |x|^k$ for some $k$.

We cannot hope to prove the existence of one-way functions without at the same time proving $P \neq NP$.

It is strongly believed that the RSA function:

$$f(x, e, p, q) = (x^e \mod pq, pq, e)$$

is a one-way function.
A function $f$ is called a **one way function** if it satisfies the following conditions:

1. $f$ is one-to-one.
2. for each $x$, $|x|^{1/k} \leq |f(x)| \leq |x|^k$ for some $k$.
3. $f$ is computable in polynomial time.

We cannot hope to prove the existence of one-way functions without at the same time proving $P \neq NP$.

It is strongly believed that the **RSA** function:

$$f(x, e, p, q) = (x^e \text{ mod } pq, pq, e)$$

is a one-way function.
One Way Functions

A function $f$ is called a one way function if it satisfies the following conditions:

1. $f$ is one-to-one.
2. for each $x$, $|x|^{1/k} \leq |f(x)| \leq |x|^k$ for some $k$.
3. $f$ is computable in polynomial time.
4. $f^{-1}$ is not computable in polynomial time.

We cannot hope to prove the existence of one-way functions without at the same time proving $P \neq NP$.

It is strongly believed that the RSA function:

$$f(x, e, p, q) = (x^e \mod pq, pq, e)$$

is a one-way function.
We have

\[ P \subseteq \text{UP} \subseteq \text{NP} \]
We have

\[ P \subseteq \text{UP} \subseteq \text{NP} \]

It seems unlikely that there are any \textit{NP}-complete problems in \textit{UP}.
We have

\[ P \subseteq \text{UP} \subseteq \text{NP} \]

It seems unlikely that there are any \text{NP}-complete problems in \text{UP}.

One-way functions exist \textit{if, and only if}, \( P \neq \text{UP} \).
Suppose that $L$ is a language that is in $\text{UP}$ but not in $\text{P}$. Let $U$ be an unambiguous machine that accepts $L$. 
Suppose that \( L \) is a language that is in UP but not in P. Let \( U \) be an unambiguous machine that accepts \( L \).

Define the function \( f_U \) by

if \( x \) is a string that encodes an accepting computation of \( U \),
then \( f_U(x) = 1y \) where \( y \) is the input string accepted by this computation.
\( f_U(x) = 0x \) otherwise.

We can prove that \( f_U \) is a one-way function.
We’ve already seen the definition $\text{SPACE}(f)$: the languages accepted by a machine which uses $O(f(n))$ tape cells on inputs of length $n$. Counting only work space.
We’ve already seen the definition $\text{SPACE}(f)$: the languages accepted by a machine which uses $O(f(n))$ tape cells on inputs of length $n$. Counting only work space.

$\text{NSPACE}(f)$ is the class of languages accepted by a nondeterministic Turing machine using at most $O(f(n))$ work space.
We’ve already seen the definition \( \text{SPACE}(f) \): the languages accepted by a machine which uses \( O(f(n)) \) tape cells on inputs of length \( n \). *Counting only work space.*

\( \text{NSPACE}(f) \) is the class of languages accepted by a *nondeterministic* Turing machine using at most \( O(f(n)) \) work space.

As we are only counting work space, it makes sense to consider bounding functions \( f \) that are less than linear.
Classes

$L = \text{SPACE}(\log n)$
Classes

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

$NL = \text{NSPACE}(\log n)$
Classes

L = SPACE(log n)

NL = NSPACE(log n)

PSPACE = \bigcup_{k=1}^{\infty} SPACE(n^k)

The class of languages decidable in polynomial space.
Classes

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

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

$\text{PSPACE} = \bigcup_{k=1}^{\infty} \text{SPACE}(n^k)$

The class of languages decidable in polynomial space.

$\text{NPSPACE} = \bigcup_{k=1}^{\infty} \text{NSPACE}(n^k)$
Classes

\[ L = \text{SPACE}(\log n) \]

\[ \text{NL} = \text{NSPACE}(\log n) \]

\[ \text{PSPACE} = \bigcup_{k=1}^{\infty} \text{SPACE}(n^k) \]

The class of languages decidable in polynomial space.

\[ \text{NPSPACE} = \bigcup_{k=1}^{\infty} \text{NSPACE}(n^k) \]

Also, define:

Classes

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

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

$\text{PSPACE} = \bigcup_{k=1}^{\infty} \text{SPACE}(n^k)$  

The class of languages decidable in polynomial space.

$\text{NPSPACE} = \bigcup_{k=1}^{\infty} \text{NSPACE}(n^k)$

Also, define:

$\text{co-NL} –$ the languages whose complements are in $NL$. 
Classes

\[ L = \text{SPACE}(\log n) \]

\[ \text{NL} = \text{NSPACE}(\log n) \]

\[ \text{PSPACE} = \bigcup_{k=1}^{\infty} \text{SPACE}(n^k) \]

\[ \text{NPSPACE} = \bigcup_{k=1}^{\infty} \text{NSPACE}(n^k) \]

The class of languages decidable in polynomial space.

Also, define:

\text{co-NL} – the languages whose complements are in \text{NL}.

\text{co-NPSPACE} – the languages whose complements are in \text{NPSPACE}. 
Inclusions

We have the following inclusions:

\[ L \subseteq NL \subseteq P \subseteq NP \subseteq \text{PSPACE} \subseteq \text{NPSPACE} \subseteq \text{EXP} \]

where \( \text{EXP} = \bigcup_{k=1}^{\infty} \text{TIME}(2^{n^k}) \)
We have the following inclusions:

\[
L \subseteq NL \subseteq P \subseteq NP \subseteq \text{PSPACE} \subseteq \text{NPSPACE} \subseteq \text{EXP}
\]

where
\[
\text{EXP} = \bigcup_{k=1}^{\infty} \text{TIME}(2^{n^k})
\]

Moreover,

\[
L \subseteq NL \cap \text{co-NL}
\]

\[
P \subseteq NP \cap \text{co-NP}
\]

\[
\text{PSPACE} \subseteq \text{NPSPACE} \cap \text{co-NPSPACE}
\]
We can scale up relations between complexity classes. For example:

\[ L = P \implies \text{PSPACE} = \text{EXP} \]

**Proof:** Let \( S \in \text{EXP} \).
We can scale up relations between complexity classes. For example:

\[ L = P \implies \text{PSPACE} = \text{EXP} \]

**Proof:** Let \( S \in \text{EXP} \).

Then \( S' = \{x01^{2|x|^{k}} : x \in S\} \in \text{P} \).
We can scale up relations between complexity classes. For example:

\[ L = P \implies \text{PSPACE} = \text{EXP} \]

**Proof:** Let \( S \in \text{EXP} \).

Then \( S' = \{ x01^{2|x|^k} : x \in S \} \in P \).

Hence, \( S' \in L \).
We can scale up relations between complexity classes. For example:

\[ L = P \implies \text{PSPACE} = \text{EXP} \]

**Proof:** Let \( S \in \text{EXP} \).

Then \( S' = \{ x01^{2|x|^k} : x \in S \} \in \text{P} \).

Hence, \( S' \in \text{L} \).

Given \( x \in S \), we can generate \( x01^{2|x|^k} \in S' \) in polynomial space.
We can scale up relations between complexity classes. For example:

$$L = P \implies \text{PSPACE} = \text{EXP}$$

**Proof:** Let $S \in \text{EXP}$. Then $S' = \{x01^{2|\cdot|_k} : x \in S\} \in P$. Hence, $S' \in L$.

Given $x \in S$, we can generate $x01^{2|\cdot|_k} \in S'$ in polynomial space.

Thus $S \in \text{PSPACE}$. 

Padding arguments
Constructible Functions
A complexity class such as $\text{TIME}(f)$ can be very unnatural, if $f$ is.
We restrict our bounding functions $f$ to be proper functions:
A complexity class such as $\text{TIME}(f)$ can be very unnatural, if $f$ is. We restrict our bounding functions $f$ to be proper functions:

**Definition**
A function $f : \mathbb{N} \rightarrow \mathbb{N}$ is constructible if:

- $f$ is non-decreasing, i.e. $f(n + 1) \geq f(n)$ for all $n$; and
A complexity class such as \( \text{TIME}(f) \) can be very unnatural, if \( f \) is.

We restrict our bounding functions \( f \) to be proper functions:

**Definition**

A function \( f : \mathbb{N} \rightarrow \mathbb{N} \) is *constructible* if:

- \( f \) is non-decreasing, i.e. \( f(n + 1) \geq f(n) \) for all \( n \); and
- there is a deterministic machine \( M \) which, on any input of length \( n \), replaces the input with the string \( 0^{f(n)} \), and \( M \) runs in time \( O(n + f(n)) \) and uses \( O(f(n)) \) work space.
Examples

All of the following functions are constructible:

- \([ \log n ]\);
All of the following functions are constructible:

- $\lceil \log n \rceil$;
- $n^2$;
Examples

All of the following functions are constructible:

- \([\log n]\);
- \(n^2\);
- \(n\);
All of the following functions are constructible:

- \([\log n]\);
- \(n^2\);
- \(n\);
- \(2^n\).
All of the following functions are constructible:

- $\lceil \log n \rceil$;
- $n^2$;
- $n$;
- $2^n$. 
All of the following functions are constructible:

- \([\log n]\);  
- \(n^2\);  
- \(n\);  
- \(2^n\).

If \(f\) and \(g\) are constructible functions, then so are  
\(f + g\), \(f \cdot g\), \(2^f\) and \(f(g)\) (this last, provided that \(f(n) > n\)).
\textbf{Using Constructible Functions}

$\text{NTIME}(f)$ can be defined as the class of those languages $L$ accepted by a \textit{nondeterministic} Turing machine $M$, such that for every $x \in L$, there is an accepting computation of $M$ on $x$ of length at most $O(f(n))$. 
**Using Constructible Functions**

\( \text{NTIME}(f) \) can be defined as the class of those languages \( L \) accepted by a *nondeterministic* Turing machine \( M \), such that for every \( x \in L \), there is an accepting computation of \( M \) on \( x \) of length at most \( O(f(n)) \).

If \( f \) is a constructible function then any language in \( \text{NTIME}(f) \) is accepted by a machine for which all computations are of length at most \( O(f(n)) \).
Using Constructible Functions

\( \text{NTIME}(f) \) can be defined as the class of those languages \( L \) accepted by a \textit{nondeterministic} Turing machine \( M \), such that for every \( x \in L \), there is an accepting computation of \( M \) on \( x \) of length at most \( O(f(n)) \).

If \( f \) is a constructible function then any language in \( \text{NTIME}(f) \) is accepted by a machine for which all computations are of length at most \( O(f(n)) \).

Also, given a Turing machine \( M \) and a constructible function \( f \), we can define a machine that simulates \( M \) for \( f(n) \) steps.
To establish the known inclusions between the main complexity classes, we prove the following, for any constructible $f$. 
Establishing Inclusions

To establish the known inclusions between the main complexity classes, we prove the following, for any constructible $f$.

- $\text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n))$;
Establishing Inclusions

To establish the known inclusions between the main complexity classes, we prove the following, for any constructible $f$.

- $\text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n))$;
- $\text{TIME}(f(n)) \subseteq \text{NTIME}(f(n))$;
To establish the known inclusions between the main complexity classes, we prove the following, for any constructible $f$.

- $\text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n))$;
- $\text{TIME}(f(n)) \subseteq \text{NTIME}(f(n))$;
- $\text{NTIME}(f(n)) \subseteq \text{SPACE}(f(n))$;
To establish the known inclusions between the main complexity classes, we prove the following, for any constructible $f$.

- $\text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n))$;
- $\text{TIME}(f(n)) \subseteq \text{NTIME}(f(n))$;
- $\text{NTIME}(f(n)) \subseteq \text{SPACE}(f(n))$;
- $\text{NSPACE}(f(n)) \subseteq \text{TIME}(k^{\log n} + f(n))$;
Establishing Inclusions

To establish the known inclusions between the main complexity classes, we prove the following, for any constructible $f$.

- $\text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n))$;
- $\text{TIME}(f(n)) \subseteq \text{NTIME}(f(n))$;
- $\text{NTIME}(f(n)) \subseteq \text{SPACE}(f(n))$;
- $\text{NSPACE}(f(n)) \subseteq \text{TIME}(k^{\log n+f(n)})$;
Establishing Inclusions

To establish the known inclusions between the main complexity classes, we prove the following, for any constructible $f$.

\begin{itemize}
  \item $\text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n))$;
  \item $\text{TIME}(f(n)) \subseteq \text{NTIME}(f(n))$;
  \item $\text{NTIME}(f(n)) \subseteq \text{SPACE}(f(n))$;
  \item $\text{NSPACE}(f(n)) \subseteq \text{TIME}(k^{\log n+f(n)})$;
\end{itemize}

The first two are straightforward from definitions.
The third is an easy simulation.
The last requires some more work.
Recall the Reachability problem: given a directed graph $G = (V, E)$ and two nodes $a, b \in V$, determine whether there is a path from $a$ to $b$ in $G$. 
Recall the Reachability problem: given a directed graph $G = (V, E)$ and two nodes $a, b \in V$, determine whether there is a path from $a$ to $b$ in $G$.

A simple search algorithm solves it:

1. mark node $a$, leaving other nodes unmarked, and initialise set $S$ to $\{a\}$;
Recall the **Reachability** problem: given a *directed* graph $G = (V, E)$ and two nodes $a, b \in V$, determine whether there is a path from $a$ to $b$ in $G$.

A simple search algorithm solves it:

1. mark node $a$, leaving other nodes unmarked, and initialise set $S$ to $\{a\}$;
2. while $S$ is not empty, choose node $i$ in $S$: remove $i$ from $S$ and for all $j$ such that there is an edge $(i, j)$ and $j$ is unmarked, mark $j$ and add $j$ to $S$;
Recall the **Reachability** problem: given a directed graph $G = (V, E)$ and two nodes $a, b \in V$, determine whether there is a path from $a$ to $b$ in $G$.

A simple search algorithm solves it:

1. mark node $a$, leaving other nodes unmarked, and initialise set $S$ to $\{a\}$;
2. while $S$ is not empty, choose node $i$ in $S$: remove $i$ from $S$ and for all $j$ such that there is an edge $(i, j)$ and $j$ is unmarked, mark $j$ and add $j$ to $S$;
3. if $b$ is marked, accept else reject.
We can use the $O(n^2)$ algorithm for Reachability to show that:

\[ \text{NSPACE}(f(n)) \subseteq \text{TIME}(k^{\log n} + f(n)) \]

for some constant $k$. 
We can use the $O(n^2)$ algorithm for Reachability to show that:

$$\text{NSPACE}(f(n)) \subseteq \text{TIME}(k \log n + f(n))$$

for some constant $k$.

Let $M$ be a nondeterministic machine working in space bounds $f(n)$. For any input $x$ of length $n$, there is a constant $c$ (depending on the number of states and alphabet of $M$) such that the total number of possible configurations of $M$ within space bounds $f(n)$ is bounded by $n \cdot c^f(n)$.

Here, $c^f(n)$ represents the number of different possible contents of the work space, and $n$ different head positions on the input.
Questions?