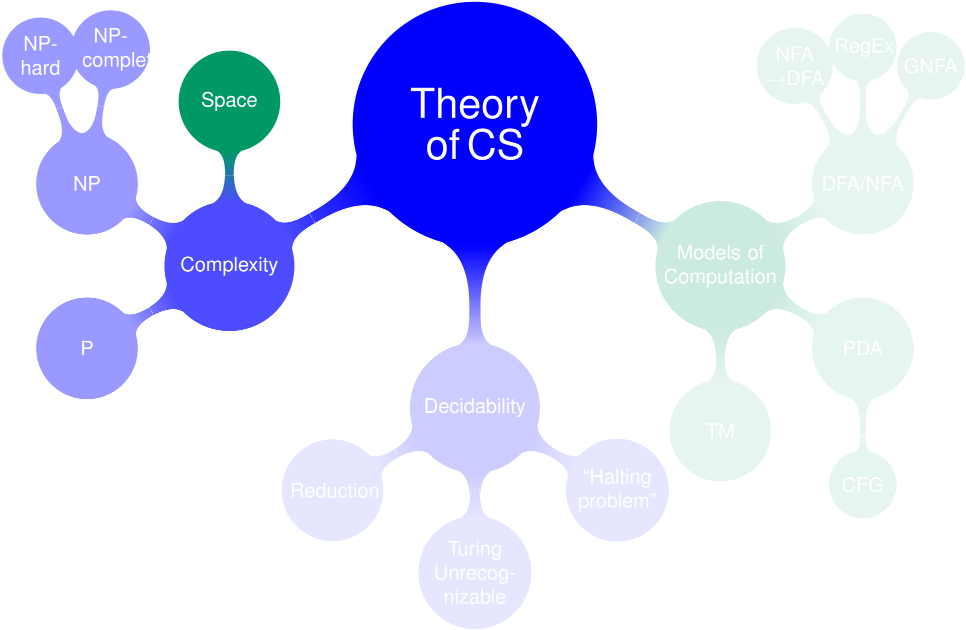
Space Complexity

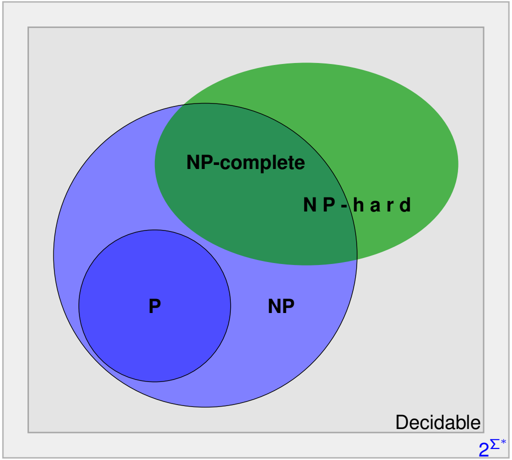
Dr Kamal Bentahar

School of Computing, Electronics and Mathematics
Coventry University

Lecture 11


Theory of CS
Complexity
Space
NP
NP-hard
NP-complet
P
Decidability
Reduction
Turing Unrecog-nizable
"Halting problem"
Models of Computation
DFA/NFA
NFA →DFA
RegEx
GNFA
PDA
CFG
TM

Last 2 lectures. . .



Space complexity

We also want to measure the amount of **memory** used by a computation.

Space complexity

(TM that always halts)

The **space complexity** of a decider $\mathcal{M}$ is the maximum number of tape cells $m(n)$ that $\mathcal{M}$ scans on any input of length n.

We say that $\mathcal{M}$ "**runs in space** $m(n)$" if its space-complexity is $m(n)$.

If $\mathcal{M}$ is non-deterministic then we measure the maximum used on any branch of its computation.

Space-complexity classes: SPACE and NSPACE

Let $m : \mathbb{N} \to \mathbb{R}^+$ be a function.

Definitions

$$
\begin{aligned}
SPACE(m(n)) &= \{L \mid L \text{ is a language decided by an } O(m(n)) \text{ space } \textbf{DTM}\} \\
NSPACE(m(n)) &= \{L \mid L \text{ is a language decided by an } O(m(n)) \text{ space } \textbf{NDTM}\}
\end{aligned}
$$

- **DTM**: Deterministic Turing Machine.
- **NDTM**: Nondeterministic Turing Machine.

If $m(n)$ is polynomial, then we call:

- $SPACE(m(n))$: **Polynomial space** or **polyspace** for short.
- $NSPACE(m(n))$: **No-deterministic polyspace.**

Example (Computing the space cost)

Consider the following decider for SAT:

On input $\langle \phi \rangle$, where ϕ is a Boolean formula with k variables $x_1, \ldots, x_k$:

1. For each truth assignment of the variables $x_1, \ldots, x_k$ of ϕ:
2. Evaluate ϕ on the current assignment.
3. If ϕ ever evaluates to *true* then *accept*; otherwise *reject*.

Let us estimate the space cost:

- Each iteration can reuse the same memory.
- Storing the current truth assignment requires k tape cells.
- So the total space needed is only $O(k)$.

We need to find the total cost as a function of $n = |\langle \phi \rangle|$, the length of the input. Since we must have $k \leq n$, then space cost is $O(k) = O(n)$.

Savitch Theorem

Savitch's Theorem

For any function $m : \mathbb{N} \to \mathbb{R}^+$, where $m(n) \geq n$,

$$NSPACE(m(n)) \subseteq SPACE(m^2(n))$$

This is really surprising!

When simulating NDTMs using DTMs:

- **Time complexity** seems to increase exponentially...
- **Space complexity** increases quadratically only!

This is because we can **reuse** space, whereas we cannot reuse time!

PSPACE vs NPSPACE

Definitions

PSPACE: class of languages that are decidable in **polyspace** on a DTM

$$\textbf{PSPACE} = SPACE(1) \cup SPACE(n) \cup SPACE(n^2) \cup \cdots$$

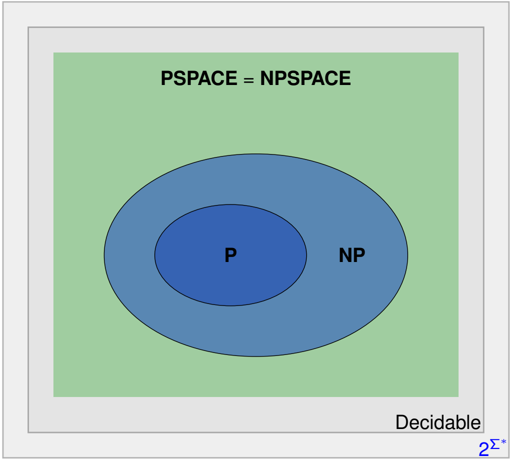
NPSPACE: class of languages that are decidable in **polyspace** on a NDTM

$$\textbf{NPSPACE} = NSPACE(1) \cup NSPACE(n) \cup NSPACE(n^2) \cup \cdots$$

By Savitch theorem, we have the surprising result:

$$\textbf{PSPACE} = \textbf{NPSPACE}$$

$\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE}$



Logarithmic space

In applications such as processing "big data" we really care about the "extra space" needed.

We model this scenario as follows:

We use a 2-tape TM:

1. The input is read-only on the first tape.
2. We measure the **extra space** used for working on the second tape.

We then define two **logarithmic space** complexity classes:

L: set of problems decidable in $O(\log n)$ space on a DTM.

NL: set of problems decidable in $O(\log n)$ space on a NDTM.

Encoding numbers

In general, given a number n, we can represent it in two ways:

- **Unary.** We would need n symbols. For example, $7_{10} = ||||||| _{unary}$.
- **Positional number system.** For example, $1000_{10} = 1111101000_2$. Using base b costs about $\log_b n$ which is $\log_2 n / \log_2 n = O(\log_2 n)$ so we just write $O(\log n)$ without specifying a base.

Example ($A = \{w \mid w = a^i b^i \quad \text{for} \quad i \geq 0\}$)

Let $n = |w|$ be the size of the input.
DTM specification:

1. Check the input is of the form a^*b^*. — No extra space is needed.
2. Keep a counter in binary to count a's. — $O(\log n)$ bits.
3. Keep a counter in binary to count b's. — $O(\log n)$ bits.
4. Check if the two counters are equal. — No extra space is needed.

Total space cost: $O(\log n)$. So, $A \in \mathbf{L}$

How do these classes compare to each other?

Define

$$\mathbf{EXPTIME} = TIME(2^n) \cup TIME(2^{n^2}) \cup TIME(2^{n^3}) \cup \cdots$$

$$\mathbf{EXPSPACE} = SPACE(2^n) \cup SPACE(2^{n^2}) \cup SPACE(2^{n^3}) \cup \cdots$$

We currently know that

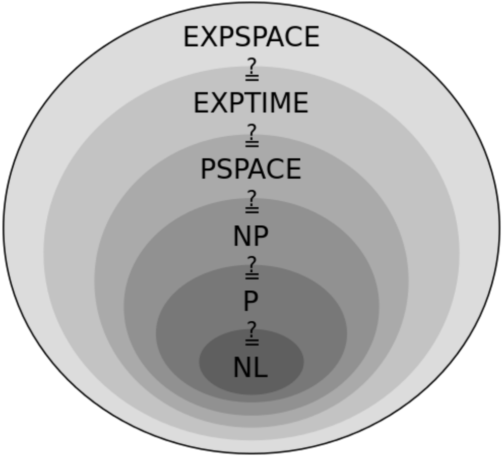
$$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXPTIME} \subseteq \mathbf{EXPSPACE}$$

We also know that

$$\mathbf{P} \neq \mathbf{EXPTIME}$$

$$\mathbf{L} \neq \mathbf{PSPACE}$$

$$\mathbf{PSPACE} \neq \mathbf{EXPSPACE}$$


EXPSPACE
?
=
EXPTIME
?
=
PSPACE
?
=
NP
?
=
P
?
=
NL

The Extended Chomsky Hierarchy