Non-Deterministic Space
Non-deterministic Machines

Recall that if $NT$ is a non-deterministic Turing Machine, then $NT(x)$ denotes the tree of configurations which can be entered with input $x$, and $NT$ accepts $x$ if there is some accepting path in $NT(x)$.

**Definition**

The space complexity of a non-deterministic Turing Machine $NT$ is the function $\text{NSpace}_{NT}$ such that $\text{NSpace}_{NT}(x)$ is the minimal number of cells visited in an accepting path of $NT(x)$ if there is one, otherwise it is the minimal number of cells in the rejecting paths.

(If not all paths of $NT(x)$ halt, then $\text{NSpace}_{NT}(x)$ is undefined)
Nondeterministic Space Complexity

**Definition**

For any function $f$, we say that the nondeterministic space complexity of a decidable language $L$ is in $O(f)$ if there exists a nondeterministic Turing Machine $NT$ which decides $L$, and constants $n_0$ and $c$ such that for all inputs $x$ with $|x| > n_0$

$$\text{NSpace}_{NT}(x) \leq cf(|x|)$$

**Definition**

The nondeterministic space complexity class $\text{NSPACE}[f]$ is defined to be the class of all languages with nondeterministic space complexity in $O(f)$
Definition of \( \text{NPSPACE} \)

\[
\text{NPSPACE} = \bigcup_{k \geq 0} \text{NSPACE}[n^k]
\]
Savitch’s Theorem

Unlike time, it can easily be shown that non-determinism does not reduce the space requirements very much:

**Theorem (Savitch)**
If \( s(n) \geq \log n \), then

\[
\text{NSPACE}[s] \subseteq \text{SPACE}[s^2]
\]

**Corollary**

\[ \text{PSPACE} = \text{NPSPACE} \]
Proof (for $s(n) \geq n$)

- Let $L$ be a language in $\text{NSPACE}[s]$.
- Let $NT$ be a non-deterministic Turing Machine that decides $L$ with space complexity $s$.
- Choose an encoding for the computation $NT(x)$ that uses $ks(|x|)$ symbols for each configuration.
- Let $C_0$ be the initial configuration, and $C_a$ be the accepting configuration.
- Define a Boolean function $reach(C,C',j)$ which is true if and only if configuration $C'$ can be reached from configuration $C$ in at most $2^j$ steps.
- To decide whether or not $x \in L$ we must determine whether or not $reach(C_0,C_a,ks(|x|))$ is true.
We can calculate \( \text{reach}(C_0, C_a, ks(|x|)) \) in \( O(s(|x|)^2) \) space, using a divide-and-conquer algorithm:

\[
\text{reach}(C, C', j)
\]

1. If \( j = 0 \) then if \( C = C' \), or \( C' \) can be reached from \( C \) in one step, then return \text{true}, else return \text{false}.

2. For each configuration \( C'' \), if \( \text{reach}(C, C'', j-1) \) and \( \text{reach}(C'', C', j-1) \), then return \text{true}.

3. Return \text{false}

The depth of recursion is \( O(s(|x|)) \) and each recursive call requires \( O(s(|x|)) \) space for the parameters.
Logarithmic Space

Since polynomial space is so powerful, it is natural to consider more restricted space complexity classes.

Even linear space is enough to solve Satisfiability

**Definition**

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

\[ \text{NL} = \text{NSPACE}[\log n] \]
Problems in L and NL

What sort of problems are in L and NL?

In logarithmic space we can store:

- a fixed number of counters (up to length of input)
- a fixed number of pointers to positions in the input string

Therefore in deterministic log-space we can solve problems that require a fixed number of counters and/or pointers for solving; in non-deterministic log-space we can solve problems that require a fixed number of counters/pointers for verifying a solution.
Examples (L)

Palindromes:

We need to keep two counter

\[ \mathcal{L} = \{0^k 1^k \mid k \in \mathbb{N} \} \]

First count the number of 0s, then count 1s, subtracting from the previous number one by one. If the result is 0, accept; otherwise, reject.

Brackets (if brackets in an expression positioned correctly):

We need only a counter of brackets currently open. If this counter gets negative, reject; otherwise accept if and only if the last value of the counter is zero.
Examples (NL)

The first problem defined on this course was Reachability\(^1\)

This can be solved by the following non-deterministic algorithm:

- Define a counter and initialize it to the number of vertices in the graph
- Define a pointer to hold the "current vertex" and initialize it to the start vertex
- While the counter is non-zero
  - If the current vertex equals the target vertex, return yes
  - Non-deterministically choose a vertex which is connected to the current vertex
  - Update the pointer to this vertex and decrement the counter
- Return no

\(^1\)Also known as Path