Relations between Complexity Classes

K. Subramani\textsuperscript{1}

\textsuperscript{1}Lane Department of Computer Science and Electrical Engineering
West Virginia University

The Reachability Method
1. The Reachability Method
   - Some basic theorems
   - Non-deterministic Space
Outline

1. The Reachability Method
   - Some basic theorems
   - Non-deterministic Space

Subramani

Complexity Classes
The Reachability Method
Some basic theorems
Non-deterministic Space

Some basic theorems

Theorem

Suppose that $f(n)$ is a proper complexity function. Then:

(i) $\text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n))$ and $\text{TIME}(f(n)) \subseteq \text{NTIME}(f(n))$.

(ii) $\text{NTIME}(f(n)) \subseteq \text{SPACE}(f(n))$.

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

Proof.

(i) and (ii) are trivial. For (iii), assume that we are given a $k$-string NDTM $M$ with input and output that decides $L$ in space $f(n)$. A configuration of $M$ can be described as $(q, i, w_2, u_2, \ldots, w_{k-1}, u_{k-1})$, where $0 \leq i \leq n$ marks a position in the input string. Total number of configurations $= |K| \times (n + 1) \times |\Sigma|^{(2k-2)f(n)} = nc_1^{f(n)} = c_1^{f(n)+\log n}$.

Create the configuration graph $G(M, x)$ on input $x$; vertices are configurations and there exists an edge from the vertex representing $C_1$ to the vertex representing $C_2$ if and only if $C_1 \rightarrow_M C_2$.

$x \in L$ if and only if there is a path from $C_0 = (s, 0, >, \epsilon, \ldots, \epsilon)$ to some $C = (\text{"yes"}, \ldots, \epsilon)$. $\square$
The Reachability Method

Some basic theorems

Non-deterministic Space

**Theorem**

Suppose that $f(n)$ is a proper complexity function. Then:

(i) $\text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n))$ and $\text{TIME}(f(n)) \subseteq \text{NTIME}(f(n))$.

(ii) $\text{NTIME}(f(n)) \subseteq \text{SPACE}(f(n))$.

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

**Proof.**

(i) and (ii) are trivial. For (iii), assume that we are given a $k$-string NDTM $M$ with input and output that decides $L$ in space $f(n)$. A configuration of $M$ can be described as $(q, i, w_2, u_2, \ldots, w_{k-1}, u_{k-1})$, where $0 \leq i \leq n$ marks a position in the input string. Total number of configurations $= |K| \times (n + 1) \times |\Sigma|^{(2k-2)f(n)} = nc_1^{f(n)} = c_1^{f(n)+\log n}$.

Create the configuration graph $G(M, x)$ on input $x$; vertices are configurations and there exists an edge from the vertex representing $C_1$ to the vertex representing $C_2$ if and only if $C_1 \rightarrow_M C_2$.

$x \in L$ if and only if there is a path from $C_0 = (s, 0, \triangleright, \epsilon, \ldots, \epsilon)$ to some $C = (\text{“yes”}, \ldots)$.
Theorem

Suppose that \( f(n) \) is a proper complexity function. Then:

(i) \( \text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n)) \) and \( \text{TIME}(f(n)) \subseteq \text{NTIME}(f(n)) \).

(ii) \( \text{NTIME}(f(n)) \subseteq \text{SPACE}(f(n)) \).

(iii) \( \text{NSPACE}(f(n)) \subseteq \text{TIME}(k^{\log n + f(n)}) \).

Proof.

(i) and (ii) are trivial. For (iii), assume that we are given a \( k \)-string NDTM \( M \) with input and output that decides \( L \) in space \( f(n) \). A configuration of \( M \) can be described as \( (q, i, w_2, u_2, \ldots, w_{k-1}, u_{k-1}) \), where \( 0 \leq i \leq n \) marks a position in the input string. Total number of configurations = \( |K| \times (n + 1) \times |\Sigma|^{(2k-2)f(n)} = nc_1^{f(n)} = c_1^{f(n)+\log n} \).

Create the configuration graph \( G(M, x) \) on input \( x \); vertices are configurations and there exists an edge from the vertex representing \( C_1 \) to the vertex representing \( C_2 \) if and only if \( C_1 \rightarrow_M C_2 \).

\( x \in L \) if and only if there is a path from \( C_0 = (s, 0, \triangleright, \epsilon, \ldots, \epsilon) \) to some \( C = ("yes", \ldots, \epsilon) \).
The Reachability Method

Some basic theorems

Non-deterministic Space

**Theorem**

Suppose that $f(n)$ is a proper complexity function. Then:

(i) $\text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n))$ and $\text{TIME}(f(n)) \subseteq \text{NTIME}(f(n))$.

(ii) $\text{NTIME}(f(n)) \subseteq \text{SPACE}(f(n))$.

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

**Proof.**

(i) and (ii) are trivial. For (iii), assume that we are given a $k$-string NDTM $M$ with input and output that decides $L$ in space $f(n)$. A configuration of $M$ can be described as $(q, i, w_2, u_2, \ldots, w_{k-1}, u_{k-1})$, where $0 \leq i \leq n$ marks a position in the input string. Total number of configurations $= |K| \times (n + 1) \times |\Sigma|^{(2k-2)f(n)} = nc_1^{f(n)} = c_1^{f(n)+\log n}$.

Create the configuration graph $G(M, x)$ on input $x$; vertices are configurations and there exists an edge from the vertex representing $C_1$ to the vertex representing $C_2$ if and only if $C_1 \rightarrow_M C_2$.

$x \in L$ if and only if there is a path from $C_0 = (s, 0, \triangleright, \epsilon, \ldots, \epsilon)$ to some $C = (\text{"yes"}, \ldots, \text{.)}$.
The Reachability Method

Some basic theorems

**Theorem**

Suppose that $f(n)$ is a proper complexity function. Then:

(i) $\text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n))$ and $\text{TIME}(f(n)) \subseteq \text{NTIME}(f(n))$.

(ii) $\text{NTIME}(f(n)) \subseteq \text{SPACE}(f(n))$.

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

**Proof.**

(i) and (ii) are trivial. For (iii), assume that we are given a $k$-string NDTM $M$ with input and output that decides $L$ in space $f(n)$. A configuration of $M$ can be described as $(q, i, w_2, u_2, \ldots, w_{k-1}, u_{k-1})$, where $0 \leq i \leq n$ marks a position in the input string. Total number of configurations $= |K| \times (n + 1) \times |\Sigma|^{(2k-2)f(n)} = n c_1^f(n) = c_1^{f(n)+\log n}$.

Create the configuration graph $G(M, x)$ on input $x$; vertices are configurations and there exists an edge from the vertex representing $C_1$ to the vertex representing $C_2$ if and only if $C_1 \rightarrow_M C_2$.

$x \in L$ if and only if there is a path from $C_0 = (s, 0, \triangleright, \epsilon, \ldots, \epsilon)$ to some $C = ("yes", \ldots, \ldots)$. \qed
Some basic theorems

**Theorem**

Suppose that $f(n)$ is a proper complexity function. Then:

(i) $\text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n))$ and $\text{TIME}(f(n)) \subseteq \text{NTIME}(f(n))$.

(ii) $\text{NTIME}(f(n)) \subseteq \text{SPACE}(f(n))$.

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

**Proof.**

(i) and (ii) are trivial. For (iii), assume that we are given a $k$-string NDTM $M$ with input and output that decides $L$ in space $f(n)$. A configuration of $M$ can be described as $(q, i, w_2, u_2, \ldots, w_{k-1}, u_{k-1})$, where $0 \leq i \leq n$ marks a position in the input string. Total number of configurations $= |K| \times (n + 1) \times |\Sigma|^{(2k-2)f(n)} = n c_1^{f(n)} = c_1^{f(n)+\log n}$.

Create the configuration graph $G(M, x)$ on input $x$; vertices are configurations and there exists an edge from the vertex representing $C_1$ to the vertex representing $C_2$ if and only if $C_1 \rightarrow_M C_2$.

$x \in L$ if and only if there is a path from $C_0 = (s, 0, \triangleright, \epsilon, \ldots, \epsilon)$ to some $C = (\text{"yes"}, \ldots, \epsilon)$.
Some basic theorems

**Theorem**

Suppose that \( f(n) \) is a proper complexity function. Then:

(i) \( \text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n)) \) and \( \text{TIME}(f(n)) \subseteq \text{NTIME}(f(n)) \).

(ii) \( \text{NTIME}(f(n)) \subseteq \text{SPACE}(f(n)) \).

(iii) \( \text{NSPACE}(f(n)) \subseteq \text{TIME}(k^{\log n+f(n)}) \).

**Proof.**

(i) and (ii) are trivial. For (iii), assume that we are given a \( k \)-string NDTM \( M \) with input and output that decides \( L \) in space \( f(n) \). A configuration of \( M \) can be described as 
\( (q, i, w_2, u_2, \ldots, w_{k-1}, u_{k-1}) \), where \( 0 \leq i \leq n \) marks a position in the input string. Total number of configurations = \( |K| \times (n + 1) \times |\Sigma|^{(2k-2)f(n)} = nc_1^{f(n)} = c_1^{f(n)+\log n} \).

Create the configuration graph \( G(M, x) \) on input \( x \); vertices are configurations and there exists an edge from the vertex representing \( C_1 \) to the vertex representing \( C_2 \) if and only if \( C_1 \rightarrow_M C_2 \).

\( x \in L \) if and only if there is a path from \( C_0 = (s, 0, >>, \epsilon, \ldots, \epsilon) \) to some \( C = ("yes", \ldots, \).
The Reachability Method

Some basic theorems

Non-deterministic Space

Some basic theorems

Theorem

Suppose that \( f(n) \) is a proper complexity function. Then:

(i) \( \text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n)) \) and \( \text{TIME}(f(n)) \subseteq \text{NTIME}(f(n)) \).

(ii) \( \text{NTIME}(f(n)) \subseteq \text{SPACE}(f(n)) \).

(iii) \( \text{NSPACE}(f(n)) \subseteq \text{TIME}(k^{\log n+f(n)}) \).

Proof.

(i) and (ii) are trivial. For (iii), assume that we are given a \( k \)-string NDTM \( M \) with input and output that decides \( L \) in space \( f(n) \). A configuration of \( M \) can be described as 
\((q, i, w_2, u_2, \ldots, w_{k-1}, u_{k-1})\), where \( 0 \leq i \leq n \) marks a position in the input string. Total number of configurations = \( |K| \times (n + 1) \times |\Sigma|^{(2k-2)f(n)} = nc_1^{f(n)} = c_1^{f(n)+\log n} \).

Create the configuration graph \( G(M, x) \) on input \( x \); vertices are configurations and there exists an edge from the vertex representing \( C_1 \) to the vertex representing \( C_2 \) if and only if \( C_1 \rightarrow_M C_2 \).

\( x \in L \) if and only if there is a path from \( C_0 = (s, 0, \triangleright, \epsilon, \ldots, \epsilon) \) to some \( C = (\text{“yes”}, \ldots) \). 

Subramani

Complexity Classes
The Reachability Method

Some basic theorems

Theorem

Suppose that \( f(n) \) is a proper complexity function. Then:

(i) \( \text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n)) \) and \( \text{TIME}(f(n)) \subseteq \text{NTIME}(f(n)) \).

(ii) \( \text{NTIME}(f(n)) \subseteq \text{SPACE}(f(n)) \).

(iii) \( \text{NSPACE}(f(n)) \subseteq \text{TIME}(k^{\log n+f(n)}) \).

Proof.

(i) and (ii) are trivial. For (iii), assume that we are given a \( k \)-string NDTM \( M \) with input and output that decides \( L \) in space \( f(n) \). A configuration of \( M \) can be described as \((q, i, w_2, u_2, \ldots, w_{k-1}, u_{k-1})\), where \( 0 \leq i \leq n \) marks a position in the input string. Total number of configurations = \(|K| \times (n + 1) \times |\Sigma|^{(2k-2)f(n)} = nc_1^{f(n)} = c_1^{f(n)+\log n}\).

Create the configuration graph \( G(M, x) \) on input \( x \); vertices are configurations and there exists an edge from the vertex representing \( C_1 \) to the vertex representing \( C_2 \) if and only if \( C_1 \rightarrow_M C_2 \).

\( x \in L \) if and only if there is a path from \( C_0 = (s, 0, \triangleright, \epsilon, \ldots, \epsilon) \) to some \( C = (\text{"yes"}, \ldots) \).
Some basic theorems

Theorem

Suppose that \( f(n) \) is a proper complexity function. Then:

(i) \( \text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n)) \) and \( \text{TIME}(f(n)) \subseteq \text{NTIME}(f(n)) \).

(ii) \( \text{NTIME}(f(n)) \subseteq \text{SPACE}(f(n)) \).

(iii) \( \text{NSPACE}(f(n)) \subseteq \text{TIME}(k^{\log n+f(n)}) \).

Proof.

(i) and (ii) are trivial. For (iii), assume that we are given a \( k \)-string NDTM \( M \) with input and output that decides \( L \) in space \( f(n) \). A configuration of \( M \) can be described as \((q, i, w_2, u_2, \ldots, w_{k-1}, u_{k-1})\), where \( 0 \leq i \leq n \) marks a position in the input string. Total number of configurations = \( |K| \times (n+1) \times |\Sigma|^{(2k-2)f(n)} = nc_1^{f(n)} = c_1^{f(n)+\log n} \).

Create the configuration graph \( G(M, x) \) on input \( x \); vertices are configurations and there exists an edge from the vertex representing \( C_1 \) to the vertex representing \( C_2 \) if and only if \( C_1 \rightarrow_M C_2 \).

\( x \in L \) if and only if there is a path from \( C_0 = (s, 0, >, \epsilon, \ldots, \epsilon) \) to some \( C = ("yes", \ldots, ) \).
Some basic theorems (contd.)

Proof.

But now the problem is REACHABILITY in a graph with \(c_1^{f(n) + \log n}\) nodes. Can be accomplished in \(c_2 \cdot (c_1^{f(n) + \log n})^2 = c_2 \cdot c_1^{2 \cdot (f(n) + \log n)} = k^{f(n) + \log n}\) time using a standard reachability algorithm.

Corollary

\[ L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE. \]
Some basic theorems (contd.)

Proof.

But now the problem is \( \text{REACHABILITY} \) in a graph with \( c_1^{f(n) + \log n} \) nodes. Can be accomplished in 
\[
c_2 \cdot (c_1^{f(n) + \log n})^2 = c_2 \cdot c_1^{2 \cdot (f(n) + \log n)} = k^{f(n) + \log n}
\]
time using a standard reachability algorithm.

Corollary

\( L \subseteq NL \subseteq P \subseteq NP \subseteq \text{PSPACE}. \)
Some basic theorems (contd.)

Proof.

But now the problem is **REACHABILITY** in a graph with $c_1^{f(n)+\log n}$ nodes. Can be accomplished in $c_2 \cdot (c_1^{f(n)+\log n})^2 = c_2 \cdot c_1^{2 \cdot (f(n)+\log n)} = k^{f(n)+\log n}$ time using a standard reachability algorithm.

Corollary

$L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE$. 
The Reachability Method

Some basic theorems (contd.)

Proof.

But now the problem is \( \text{REACHABILITY} \) in a graph with \( c_1^{f(n) + \log n} \) nodes. Can be accomplished in \( c_2 \cdot (c_1^{f(n) + \log n})^2 = c_2 \cdot c_1^{2 \cdot (f(n) + \log n)} = k^{f(n) + \log n} \) time using a standard reachability algorithm.

Corollary

\[ L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE. \]
Outline

1. The Reachability Method
   - Some basic theorems
   - Non-deterministic Space
Savitch’s Theorem

Theorem

\[ \text{REACHABILITY} \in \text{SPACE}(\log^2 n). \]

Proof.

Let \( G \) be a graph with \( n \) nodes and \( x, y \in G \). \( \text{PATH}(x, y, i) \) is \( \text{true} \), if there is a path of length at most \( 2^i \) from \( x \) to \( y \) in \( G \). \( \text{REACHABILITY} \) coincides with checking whether \( \text{PATH}(x, y, \lceil \log n \rceil) \) is \( \text{true} \).

We design a 2-string Turing machine with input and output. The adjacency matrix of \( G \) is stored on the input string. The first string contains several triples with \((x, y, i)\) denoting the first triple. The second string will be used as scratch space.

Two cases to consider

1. \( i = 0 \) - Check if \((x, y)\) is an edge!
2. \( i \geq 1 \) - Implement the following recursion:
   
   \[ \text{for all nodes } z \in G, \text{ test whether } \text{PATH}(x, z, i - 1) \land \text{PATH}(z, y, i - 1). \]

Implementing the recursion in a space efficient manner - Generate all vertices \( z \), one after the other reusing space. Interpret positive and negative answers to \( \text{PATH}(x, z, i - 1) \) correctly. Stack size is at most \( \log n \) triples of size \( 3 \log n \) each. Thus, total space used is \( O(\log^2 n) \).
The Reachability Method

Savitch’s Theorem

Theorem

\textbf{REACHABILITY} \in \textbf{SPACE}(\log^2 n).

Proof.

Let \( G \) be a graph with \( n \) nodes and \( x, y \in G \). \textbf{PATH}(x, y, i) is \textbf{true}, if there is a path of length at most \( 2^i \) from \( x \) to \( y \) in \( G \). \textbf{REACHABILITY} coincides with checking whether \( \text{PATH}(x, y, \lceil \log n \rceil) \) is \textbf{true}.

We design a 2-string Turing machine with input and output. The adjacency matrix of \( G \) is stored on the input string. The first string contains several triples with \((x, y, i)\) denoting the first triple. The second string will be used as scratch space.

Two cases to consider

(i) \( i = 0 \) - Check if \((x, y)\) is an edge!

(ii) \( i \geq 1 \) - Implement the following recursion:

\[ \text{for all nodes } z \in G, \text{ test whether } \text{PATH}(x, z, i - 1) \wedge \text{PATH}(z, y, i - 1). \]

Implementing the recursion in a space efficient manner - Generate all vertices \( z \), one after the other reusing space. Interpret positive and negative answers to \( \text{PATH}(x, z, i - 1) \) correctly. Stack size is at most \( \log n \) triples of size \( 3 \log n \) each. Thus, total space used is \( O(\log^2 n) \).
Savitch’s Theorem

**Theorem**

\[ \text{REACHABILITY} \in \text{SPACE}(\log^2 n). \]

**Proof.**

Let \( G \) be a graph with \( n \) nodes and \( x, y \in G \). \( \text{PATH}(x, y, i) \) is **true**, if there is a path of length at most \( 2^i \) from \( x \) to \( y \) in \( G \). \( \text{REACHABILITY} \) coincides with checking whether \( \text{PATH}(x, y, \lceil \log n \rceil) \) is **true**.

We design a 2-string Turing machine with input and output. The adjacency matrix of \( G \) is stored on the input string. The first string contains several triples with \((x, y, i)\) denoting the first triple. The second string will be used as scratch space.

Two cases to consider

(i) \( i = 0 \) - Check if \((x, y)\) is an edge!

(ii) \( i \geq 1 \) - Implement the following recursion:

\[ \text{for all nodes } z \in G, \text{ test whether } \text{PATH}(x, z, i − 1) \land \text{PATH}(z, y, i − 1). \]

Implementing the recursion in a space efficient manner - Generate all vertices \( z \), one after the other reusing space. Interpret positive and negative answers to \( \text{PATH}(x, z, i − 1) \) correctly. Stack size is at most \( \log n \) triples of size \( 3 \log n \) each. Thus, total space used is \( O(\log^2 n) \).
The Reachability Method

Savitch’s Theorem

Theorem

\[ \text{REACHABILITY} \in \text{SPACE}(\log^2 n). \]

Proof.

Let \( G \) be a graph with \( n \) nodes and \( x, y \in G \). \( \text{PATH}(x, y, i) \) is true, if there is a path of length at most \( 2^i \) from \( x \) to \( y \) in \( G \). \( \text{REACHABILITY} \) coincides with checking whether \( \text{PATH}(x, y, \lceil \log n \rceil) \) is true.

We design a 2-string Turing machine with input and output. The adjacency matrix of \( G \) is stored on the input string. The first string contains several triples with \((x, y, i)\) denoting the first triple. The second string will be used as scratch space.

Two cases to consider:

(i) \( i = 0 \) - Check if \((x, y)\) is an edge!

(ii) \( i \geq 1 \) - Implement the following recursion:

\[ \text{for all nodes } z \in G, \text{ test whether } \text{PATH}(x, z, i - 1) \land \text{PATH}(z, y, i - 1). \]

Implementing the recursion in a space efficient manner - Generate all vertices \( z \), one after the other reusing space. Interpret positive and negative answers to \( \text{PATH}(x, z, i - 1) \) correctly. Stack size is at most \( \log n \) triples of size \( 3 \log n \) each. Thus, total space used is \( O(\log^2 n) \).
Savitch’s Theorem

Theorem

\( \text{REACHABILITY} \in \text{SPACE}(\log^2 n). \)

Proof.

Let \( G \) be a graph with \( n \) nodes and \( x, y \in G \). \( \text{PATH}(x, y, i) \) is \textbf{true}, if there is a path of length at most \( 2^i \) from \( x \) to \( y \) in \( G \). \( \text{REACHABILITY} \) coincides with checking whether \( \text{PATH}(x, y, \lceil \log n \rceil) \) is \textbf{true}.

We design a 2-string Turing machine with input and output. The adjacency matrix of \( G \) is stored on the input string. The first string contains several triples with \( (x, y, i) \) denoting the first triple. The second string will be used as scratch space.

Two cases to consider

(i) \( i = 0 \) - Check if \( (x, y) \) is an edge!

(ii) \( i \geq 1 \) - Implement the following recursion:

\[ \text{for all nodes } z \in G, \text{ test whether } \text{PATH}(x, z, i - 1) \land \text{PATH}(z, y, i - 1). \]

Implementing the recursion in a space efficient manner - Generate all vertices \( z \), one after the other reusing space. Interpret positive and negative answers to \( \text{PATH}(x, z, i - 1) \) correctly. Stack size is at most \( \log n \) triples of size \( 3 \log n \) each. Thus, total space used is \( O(\log^2 n) \).
Savitch’s Theorem

**Theorem**

\[ \text{REACHABILITY} \in \text{SPACE}(\log^2 n). \]

**Proof.**

Let \( G \) be a graph with \( n \) nodes and \( x, y \in G \). \( \text{PATH}(x, y, i) \) is \textbf{true}, if there is a path of length at most \( 2^i \) from \( x \) to \( y \) in \( G \). \( \text{REACHABILITY} \) coincides with checking whether \( \text{PATH}(x, y, \lceil \log n \rceil) \) is \textbf{true}.

We design a 2-string Turing machine with input and output. The adjacency matrix of \( G \) is stored on the input string. The first string contains several triples with \((x, y, i)\) denoting the first triple. The second string will be used as scratch space.

Two cases to consider

(i) \( i = 0 \) - Check if \((x, y)\) is an edge!

(ii) \( i \geq 1 \) - Implement the following recursion:

\[ \text{for all nodes } z \in G, \text{ test whether } \text{PATH}(x, z, i - 1) \land \text{PATH}(z, y, i - 1). \]

Implementing the recursion in a space efficient manner - Generate all vertices \( z \), one after the other reusing space. Interpret positive and negative answers to \( \text{PATH}(x, z, i - 1) \) correctly. Stack size is at most \( \log n \) triples of size \( 3 \log n \) each. Thus, total space used is \( O(\log^2 n) \).
The Reachability Method

Savitch’s Theorem

**Theorem**

\[ \text{REACHABILITY} \in \text{SPACE}(\log^2 n). \]

**Proof.**

Let \( G \) be a graph with \( n \) nodes and \( x, y \in G \). \( \text{PATH}(x, y, i) \) is true, if there is a path of length at most \( 2^i \) from \( x \) to \( y \) in \( G \).\text{REACHABILITY} coincides with checking whether \( \text{PATH}(x, y, \lceil \log n \rceil) \) is true.

We design a 2-string Turing machine with input and output. The adjacency matrix of \( G \) is stored on the input string. The first string contains several triples with \((x, y, i)\) denoting the first triple. The second string will be used as scratch space.

Two cases to consider

(i) \( i = 0 \) - Check if \((x, y)\) is an edge!

(ii) \( i \geq 1 \) - Implement the following recursion:

\[
\text{for all nodes } z \in G, \text{ test whether } \text{PATH}(x, z, i - 1) \land \text{PATH}(z, y, i - 1).
\]

Implementing the recursion in a space efficient manner - Generate all vertices \( z \), one after the other reusing space. Interpret positive and negative answers to \( \text{PATH}(x, z, i - 1) \) correctly. Stack size is at most \( \log n \) triples of size \( 3 \log n \) each. Thus, total space used is \( O(\log^2 n) \).
The Reachability Method

Savitch’s Theorem

**Theorem**

\[ \text{REACHABILITY} \in \text{SPACE}(\log^2 n). \]

**Proof.**

Let \( G \) be a graph with \( n \) nodes and \( x, y \in G \). \( \text{PATH}(x, y, i) \) is true, if there is a path of length at most \( 2^i \) from \( x \) to \( y \) in \( G \). \( \text{REACHABILITY} \) coincides with checking whether \( \text{PATH}(x, y, \lceil \log n \rceil) \) is true.

We design a 2-string Turing machine with input and output. The adjacency matrix of \( G \) is stored on the input string. The first string contains several triples with \((x, y, i)\) denoting the first triple. The second string will be used as scratch space.

Two cases to consider

(i) \( i = 0 \) - Check if \((x, y)\) is an edge!

(ii) \( i \geq 1 \) - Implement the following recursion:

\[ \text{for all nodes } z \in G, \text{ test whether } \text{PATH}(x, z, i - 1) \land \text{PATH}(z, y, i - 1). \]

Implementing the recursion in a space efficient manner - Generate all vertices \( z \), one after the other reusing space. Interpret positive and negative answers to \( \text{PATH}(x, z, i - 1) \) correctly. Stack size is at most \( \log n \) triples of size \( 3 \log n \) each. Thus, total space used is \( O(\log^2 n) \).
Savitch’s Theorem

**Theorem**

\[ \text{REACHABILITY} \in \text{SPACE}(\log^2 n). \]

**Proof.**

Let \( G \) be a graph with \( n \) nodes and \( x, y \in G \). \( \text{PATH}(x, y, i) \) is \textbf{true}, if there is a path of length at most \( 2^i \) from \( x \) to \( y \) in \( G \). \( \text{REACHABILITY} \) coincides with checking whether \( \text{PATH}(x, y, \lceil \log n \rceil) \) is \textbf{true}.

We design a 2-string Turing machine with input and output. The adjacency matrix of \( G \) is stored on the input string. The first string contains several triples with \((x, y, i)\) denoting the first triple. The second string will be used as scratch space.

Two cases to consider

(i) \( i = 0 \) - Check if \((x, y)\) is an edge!

(ii) \( i \geq 1 \) - Implement the following recursion:

\[
\text{for all nodes } z \in G, \text{ test whether } \text{PATH}(x, z, i - 1) \land \text{PATH}(z, y, i - 1).
\]

Implementing the recursion in a space efficient manner - Generate all vertices \( z \), one after the other reusing space. Interpret positive and negative answers to \( \text{PATH}(x, z, i - 1) \) correctly. Stack size is at most \( \log n \) triples of size \( 3 \log n \) each. Thus, total space used is \( O(\log^2 n) \).
The Reachability Method

Some basic theorems

Non-deterministic Space

Savitch’s Theorem

Theorem

\textsc{reachability} \in \textsc{space}(\log^2 n).

Proof.

Let \( G \) be a graph with \( n \) nodes and \( x, y \in G \). \textsc{path}(x, y, i) is true, if there is a path of length at most \( 2^i \) from \( x \) to \( y \) in \( G \). \textsc{reachability} coincides with checking whether \textsc{path}(x, y, \lceil \log n \rceil) is true.

We design a 2-string Turing machine with input and output. The adjacency matrix of \( G \) is stored on the input string. The first string contains several triples with \((x, y, i)\) denoting the first triple. The second string will be used as scratch space.

Two cases to consider

(i) \( i = 0 \) - Check if \((x, y)\) is an edge!

(ii) \( i \geq 1 \) - Implement the following recursion:

\[
\text{for all nodes } z \in G, \text{ test whether } \text{path}(x, z, i - 1) \land \text{path}(z, y, i - 1).
\]

Implementing the recursion in a space efficient manner - Generate all vertices \( z \), one after the other reusing space. Interpret positive and negative answers to \( \text{path}(x, z, i - 1) \) correctly. Stack size is at most \( \log n \) triples of size \( 3 \log n \) each. Thus, total space used is \( O(\log^2 n) \).
The Reachability Method

Savitch’s Theorem

**Theorem**

\[
\text{REACHABILITY} \in \text{SPACE}(\log^2 n).
\]

**Proof.**

Let \( G \) be a graph with \( n \) nodes and \( x, y \in G \). \( \text{PATH}(x, y, i) \) is \textbf{true}, if there is a path of length at most \( 2^i \) from \( x \) to \( y \) in \( G \). \( \text{REACHABILITY} \) coincides with checking whether \( \text{PATH}(x, y, \lceil \log n \rceil) \) is \textbf{true}.

We design a 2-string Turing machine with input and output. The adjacency matrix of \( G \) is stored on the input string. The first string contains several triples with \((x, y, i)\) denoting the first triple. The second string will be used as scratch space.

Two cases to consider

(i) \( i = 0 \) - Check if \((x, y)\) is an edge!

(ii) \( i \geq 1 \) - Implement the following recursion:

\[
\text{for all nodes } z \in G, \text{ test whether } \text{PATH}(x, z, i - 1) \land \text{PATH}(z, y, i - 1).
\]

Implementing the recursion in a space efficient manner - Generate all vertices \( z \), one after the other reusing space. Interpret positive and negative answers to \( \text{PATH}(x, z, i - 1) \) correctly. Stack size is at most \( \log n \) triples of size \( 3 \log n \) each. Thus, total space used is \( O(\log^2 n) \).
Savitch’s Theorem

**Theorem**

\[ \text{REACHABILITY} \in \text{SPACE}(\log^2 n). \]

**Proof.**

Let \( G \) be a graph with \( n \) nodes and \( x, y \in G \). \( \text{PATH}(x, y, i) \) is true, if there is a path of length at most \( 2^i \) from \( x \) to \( y \) in \( G \). \( \text{REACHABILITY} \) coincides with checking whether \( \text{PATH}(x, y, \lceil \log n \rceil) \) is true.

We design a 2-string Turing machine with input and output. The adjacency matrix of \( G \) is stored on the input string. The first string contains several triples with \( (x, y, i) \) denoting the first triple. The second string will be used as scratch space.

Two cases to consider

(i) \( i = 0 \) - Check if \( (x, y) \) is an edge!

(ii) \( i \geq 1 \) - Implement the following recursion:

\[ \text{for all nodes } z \in G, \text{ test whether } \text{PATH}(x, z, i - 1) \land \text{PATH}(z, y, i - 1). \]

Implementing the recursion in a space efficient manner - Generate all vertices \( z \), one after the other reusing space. Interpret positive and negative answers to \( \text{PATH}(x, z, i - 1) \) correctly. Stack size is at most \( \log n \) triples of size \( 3 \log n \) each. Thus, total space used is \( O(\log^2 n) \).
Savitch’s Theorem

Theorem

\( \text{REACHABILITY} \in \text{SPACE}(\log^2 n) \).

Proof.

Let \( G \) be a graph with \( n \) nodes and \( x, y \in G \). \( \text{PATH}(x, y, i) \) is true, if there is a path of length at most \( 2^i \) from \( x \) to \( y \) in \( G \). \text{REACHABILITY} coincides with checking whether \( \text{PATH}(x, y, \lceil \log n \rceil) \) is true.

We design a 2-string Turing machine with input and output. The adjacency matrix of \( G \) is stored on the input string. The first string contains several triples with \((x, y, i)\) denoting the first triple. The second string will be used as scratch space.

Two cases to consider

(i) \( i = 0 \) - Check if \((x, y)\) is an edge!

(ii) \( i \geq 1 \) - Implement the following recursion:

\[
\text{for all nodes } z \in G, \text{ test whether } \text{PATH}(x, z, i - 1) \land \text{PATH}(z, y, i - 1).
\]

Implementing the recursion in a space efficient manner - Generate all vertices \( z \), one after the other reusing space. Interpret positive and negative answers to \( \text{PATH}(x, z, i - 1) \) correctly. Stack size is at most \( \log n \) triples of size \( 3 \log n \) each. Thus, total space used is \( O(\log^2 n) \).
The Reachability Method

Savitch’s Theorem

**Theorem**

REACHABILITY ∈ SPACE(\(\log^2 n\)).

**Proof.**

Let \(G\) be a graph with \(n\) nodes and \(x, y \in G\). PATH\((x, y, i)\) is true, if there is a path of length at most \(2^i\) from \(x\) to \(y\) in \(G\). REACHABILITY coincides with checking whether PATH\((x, y, \lceil \log n \rceil)\) is true.

We design a 2-string Turing machine with input and output. The adjacency matrix of \(G\) is stored on the input string. The first string contains several triples with \((x, y, i)\) denoting the first triple. The second string will be used as scratch space.

Two cases to consider

(i) \(i = 0\) - Check if \((x, y)\) is an edge!

(ii) \(i \geq 1\) - Implement the following recursion:

\[
\text{for all nodes } z \in G, \text{ test whether } PATH(x, z, i - 1) \land PATH(z, y, i - 1).
\]

Implementing the recursion in a space efficient manner - Generate all vertices \(z\), one after the other reusing space. Interpret positive and negative answers to PATH\((x, z, i - 1)\) correctly. Stack size is at most \(\log n\) triples of size \(3 \log n\) each. Thus, total space used is \(O(\log^2 n)\).
**Savitch’s Theorem**

**Theorem**

\[ \text{REACHABILITY} \in \text{SPACE}(\log^2 n). \]

**Proof.**

Let \( G \) be a graph with \( n \) nodes and \( x, y \in G \). \( \text{PATH}(x, y, i) \) is true, if there is a path of length at most \( 2^i \) from \( x \) to \( y \) in \( G \). \( \text{REACHABILITY} \) coincides with checking whether \( \text{PATH}(x, y, \lceil \log n \rceil) \) is true.

We design a 2-string Turing machine with input and output. The adjacency matrix of \( G \) is stored on the input string. The first string contains several triples with \( (x, y, i) \) denoting the first triple. The second string will be used as scratch space.

Two cases to consider

(i) \( i = 0 \) - Check if \( (x, y) \) is an edge!

(ii) \( i \geq 1 \) - Implement the following recursion:

\[ \text{for all nodes } z \in G, \text{ test whether } \text{PATH}(x, z, i - 1) \land \text{PATH}(z, y, i - 1). \]

Implementing the recursion in a space efficient manner - Generate all vertices \( z \), one after the other reusing space. Interpret positive and negative answers to \( \text{PATH}(x, z, i - 1) \) correctly.

Stack size is at most \( \log n \) triples of size \( 3 \log n \) each. Thus, total space used is \( O(\log^2 n) \).
The Reachability Method

Savitch’s Theorem

Theorem

**REACHABILITY** ∈ **SPACE**(log² n).

Proof.

Let G be a graph with n nodes and x, y ∈ G. PATH(x, y, i) is **true**, if there is a path of length at most 2^i from x to y in G. **REACHABILITY** coincides with checking whether PATH(x, y, ⌈log n⌉) is **true**.

We design a 2-string Turing machine with input and output. The adjacency matrix of G is stored on the input string. The first string contains several triples with (x, y, i) denoting the first triple. The second string will be used as scratch space.

Two cases to consider

(i) i = 0 - Check if (x, y) is an edge!

(ii) i ≥ 1 - Implement the following recursion:

for all nodes z ∈ G, test whether PATH(x, z, i − 1) ∧ PATH(z, y, i − 1).

Implementing the recursion in a space efficient manner - Generate all vertices z, one after the other reusing space. Interpret positive and negative answers to PATH(x, z, i − 1) correctly. Stack size is at most log n triples of size 3 log n each. Thus, total space used is **O**(log² n).
The Reachability Method

Savitch’s Theorem

Theorem

\[ \text{REACHABILITY} \in \text{SPACE}(\log^2 n). \]

Proof.

Let \( G \) be a graph with \( n \) nodes and \( x, y \in G \). \( \text{PATH}(x, y, i) \) is true, if there is a path of length at most \( 2^i \) from \( x \) to \( y \) in \( G \). \( \text{REACHABILITY} \) coincides with checking whether \( \text{PATH}(x, y, \lceil \log n \rceil) \) is true.

We design a 2-string Turing machine with input and output. The adjacency matrix of \( G \) is stored on the input string. The first string contains several triples \((x, y, i)\) denoting the first triple. The second string will be used as scratch space.

Two cases to consider

(i) \( i = 0 \) - Check if \((x, y)\) is an edge!

(ii) \( i \geq 1 \) - Implement the following recursion:

\[ \text{for all nodes } z \in G, \text{ test whether } \text{PATH}(x, z, i - 1) \land \text{PATH}(z, y, i - 1). \]

Implementing the recursion in a space efficient manner - Generate all vertices \( z \), one after the other reusing space. Interpret positive and negative answers to \( \text{PATH}(x, z, i - 1) \) correctly. Stack size is at most \( \log n \) triples of size \( 3 \log n \) each. Thus, total space used is \( O(\log^2 n) \).
Corollary

\[ \text{NSPACE}(f(n)) \subseteq \text{SPACE}((f(n))^2) \text{ for any proper complexity function } f(n) \geq \log n. \]

Proof.

Given an \( f(n) \)-space bounded NDTM, simply run the previous algorithm on the configuration graph \( G(M, x) \), where \(|x| = n\). Since \( G(M, |x|) \) has at most \( c^{f(n)} \) nodes, \( O((f(n))^2) \) space suffices.

Corollary

\[ \text{PSPACE} = \text{NPSPACE}. \]
Corollary

\[ \text{NSPACE}(f(n)) \subseteq \text{SPACE}((f(n))^2) \text{ for any proper complexity function } f(n) \geq \log n. \]

Proof.

Given an \( f(n) \)-space bounded NDTM, simply run the previous algorithm on the configuration graph \( G(M, x) \), where \(|x| = n\). Since \( G(M, |x|) \) has at most \( c^{f(n)} \) nodes, \( O((f(n))^2) \) space suffices.

Corollary

\[ \text{PSPACE} = \text{NPSPACE}. \]
Corollary

\[
\text{NSPACE} f(n) \subseteq \text{SPACE}((f(n))^2) \text{ for any proper complexity function } f(n) \geq \log n.
\]

Proof.

Given an \( f(n) \)-space bounded NDTM, simply run the previous algorithm on the configuration graph \( G(M, x) \), where \( |x| = n \). Since \( G(M, |x|) \) has at most \( c^{f(n)} \) nodes, \( O((f(n))^2) \) space suffices.

Corollary

\[
\text{PSPACE} = \text{NPSPACE}.
\]
Counting the number of reachable nodes

Definition
A NDTM $M$ is said to compute a function $f$ from strings to strings, if all “yes” leaves have the output $f(x)$.

Theorem (Immerman-Szelepscényi Theorem)
Given a graph $G$ with $n$ nodes, and a node $x \in G$, the number of nodes reachable from $x$ in $G$ can be computed by a NDTM in space $\log n$.

Proof.
Let $S(i)$ denote the set of vertices that can be reached from $x$ using paths of length at most $i$. We are interested in $|S(n - 1)|$.

$\text{loop}_1: |S(0)| := 1; \text{ for } i = 1, 2, \ldots, n - 1: \text{compute } |S(k)| \text{ from } |S(k - 1)|.$
Counting the number of reachable nodes

**Definition**

A NDTM $M$ is said to compute a function $f$ from strings to strings, if all “yes” leaves have the output $f(x)$.

**Theorem (Immerman-Szelepcscényi Theorem)**

*Given a graph $G$ with $n$ nodes, and a node $x \in G$, the number of nodes reachable from $x$ in $G$ can be computed by a NDTM in space $\log n$.***

**Proof.**

Let $S(i)$ denote the set of vertices that can be reached from $x$ using paths of length at most $i$. We are interested in $|S(n - 1)|$.

`loop1 : |S(0)| := 1; for i = 1, 2, …, n - 1: compute |S(k)| from |S(k - 1)|. `
Counting the number of reachable nodes

Definition
A NDTM $M$ is said to compute a function $f$ from strings to strings, if all “yes” leaves have the output $f(x)$.

Theorem (Immerman-Szelepscényi Theorem)
Given a graph $G$ with $n$ nodes, and a node $x \in G$, the number of nodes reachable from $x$ in $G$ can be computed by a NDTM in space $\log n$.

Proof.
Let $S(i)$ denote the set of vertices that can be reached from $x$ using paths of length at most $i$. We are interested in $|S(n - 1)|$.

$\text{loop}_1 : |S(0)| := 1; \text{for } i = 1, 2, \ldots, n - 1: \text{compute } |S(k)| \text{ from } |S(k - 1)|.$
Counting the number of reachable nodes

**Definition**

A NDTM $M$ is said to compute a function $f$ from strings to strings, if all “yes” leaves have the output $f(x)$.

**Theorem (Immerman-Szelepscényi Theorem)**

*Given a graph $G$ with $n$ nodes, and a node $x \in G$, the number of nodes reachable from $x$ in $G$ can be computed by a NDTM in space $\log n$.***

**Proof.**

Let $S(i)$ denote the set of vertices that can be reached from $x$ using paths of length at most $i$. We are interested in $|S(n - 1)|$.

$\text{loop}_1: |S(0)| := 1; \text{ for } i = 1, 2, \ldots, n - 1: \text{ compute } |S(k)| \text{ from } |S(k - 1)|.$
Counting the number of reachable nodes

Definition

A NDTM $M$ is said to compute a function $f$ from strings to strings, if all “yes” leaves have the output $f(x)$.

Theorem (Immerman-Szelepscényi Theorem)

Given a graph $G$ with $n$ nodes, and a node $x \in G$, the number of nodes reachable from $x$ in $G$ can be computed by a NDTM in space $\log n$.

Proof.

Let $S(i)$ denote the set of vertices that can be reached from $x$ using paths of length at most $i$. We are interested in $|S(n - 1)|$.

loop$_1$ : $|S(0)| := 1$; for $i = 1, 2, \ldots, n - 1$: compute $|S(k)|$ from $|S(k - 1)|$. 
Counting the number of reachable nodes (contd.)

Proof.

\(\text{loop}_2: l := 0; \text{for each node } u = 1, 2, \ldots n: \text{if } u \in S(k), \text{ then } l := l + 1.\)

How to decide whether \(u \in S(k)\)?

\(\text{loop}_3: m := 0; \text{reply} = \text{false}; \text{for each node } v = 1, 2, \ldots n \text{ repeat:}\)

if \(v \in S(k - 1)\) then \(m := m + 1.\) Further, if \(G(v, u)\), then \(\text{reply} = \text{true}.\)

if at end, \(m < |S(k - 1)|, \text{ then "no", else return } \text{reply}.\)

How to check if \(v \in S(k - 1)\)?

Simple! Start at node \(x\) and guess \(k - 1\) nodes.

\(\text{loop}_4: w_0 := x. \text{for } p = 1, 2, \ldots k - 1:\)

guess a node \(w_p\) and check that \(G(w_{p-1}, w_p).\) (If not, return “no”).

if \(w_{k-1} = v\), then report \(v \in S_{k-1}\), else “no”.

---

Subramani Complexity Classes
Proof.

\(\text{loop}_2: \ l := 0; \ \text{for each node } u = 1, 2, \ldots n: \ \text{if } u \in S(k), \ \text{then } l := l + 1.\)

How to decide whether \(u \in S(k)\)?

\(\text{loop}_3: \ m := 0; \ \text{reply} = \text{false}; \ \text{for each node } v = 1, 2, \ldots n \ \text{repeat:}\)

if \(v \in S(k - 1)\) then \(m := m + 1.\) Further, if \(G(v, u)\), then \(\text{reply} = \text{true}.\)

if at end, \(m < |S(k - 1)|,\) then “no”, else return \(\text{reply}.\)

How to check if \(v \in S(k - 1)\)?

Simple! Start at node \(x\) and guess \(k - 1\) nodes.

\(\text{loop}_4: \ w_0 := x. \ \text{for } p = 1, 2, \ldots k - 1: \)

guess a node \(w_p\) and check that \(G(w_{p-1}, w_p)\). (If not, return “no”).

if \(w_{k-1} = v\), then report \(v \in S_{k-1}\), else “no”.

Subramani Complexity Classes
Counting the number of reachable nodes (contd.)

Proof.

\( \text{loop}_2 \) : \( l := 0; \) \textbf{for} each node \( u = 1, 2, \ldots n \): if \( u \in S(k) \), then \( l := l + 1 \).

How to decide whether \( u \in S(k) \)?

\( \text{loop}_3 \) : \( m := 0; \) \textit{reply} = \textbf{false}; \textbf{for} each node \( v = 1, 2, \ldots n \) repeat:

if \( v \in S(k - 1) \) then \( m := m + 1 \). Further, if \( G(v, u) \), then \textit{reply} = \textbf{true}.

if at end, \( m < |S(k - 1)| \), then “no”, else return \textit{reply}.

How to check if \( v \in S(k - 1) \)?

Simple! Start at node \( x \) and guess \( k - 1 \) nodes.

\( \text{loop}_4 \) : \( w_0 := x. \) \textbf{for} \( p = 1, 2, \ldots k - 1 \):

guess a node \( w_p \) and check that \( G(w_{p - 1}, w_p) \). (If not, return “no”).

if \( w_{k - 1} = v \), then report \( v \in S_{k - 1} \), else “no”.
Counting the number of reachable nodes (contd.)

**Proof.**

*loop*₂ : \( l := 0; \) for each node \( u = 1, 2, \ldots n \): if \( u \in S(k) \), then \( l := l + 1 \).

How to decide whether \( u \in S(k) \)?

*loop*₃ : \( m := 0; \) reply = false; for each node \( v = 1, 2, \ldots n \) repeat:

if \( v \in S(k - 1) \) then \( m := m + 1 \). Further, if \( G(v, u) \), then reply = true.

if at end, \( m < |S(k - 1)| \), then “no”, else return reply.

How to check if \( v \in S(k - 1) \)?

Simple! Start at node \( x \) and guess \( k - 1 \) nodes.

*loop*₄ : \( w_0 := x. \) for \( p = 1, 2, \ldots k - 1 \):

guess a node \( w_p \) and check that \( G(w_{p-1}, w_p) \). (If not, return “no”).

if \( w_{k-1} = v \), then report \( v \in S_{k-1} \), else “no”.

Subramani  Complexity Classes
Counting the number of reachable nodes (contd.)

Proof.

\textit{loop}_2: \; l := 0; \; \textbf{for} \; \text{each node} \; u = 1, 2, \ldots n: \; \text{if} \; u \in S(k), \; \text{then} \; l := l + 1.
How to decide whether \( u \in S(k) \)?

\textit{loop}_3: \; m := 0; \; \textit{reply} = \textbf{false}; \; \text{for} \; \text{each node} \; v = 1, 2, \ldots n \; \text{repeat:}
\text{if} \; v \in S(k - 1) \; \text{then} \; m := m + 1. \; \text{Further, if} \; G(v, u), \; \text{then} \; \textit{reply} = \textbf{true}.
\text{if} \; \text{at end}, \; m < |S(k - 1)|, \; \text{then} \; \text{“no”}, \; \text{else} \; \text{return} \; \textit{reply}.
How to check if \( v \in S(k - 1) \)?

Simple! Start at node \( x \) and guess \( k - 1 \) nodes.
\textit{loop}_4: \; w_0 := x. \; \text{for} \; p = 1, 2, \ldots k - 1:
\text{guess a node} \; w_p \; \text{and check that} \; G(w_{p - 1}, w_p). \; \text{(If not, return “no”).}
\text{if} \; w_{k - 1} = v, \; \text{then report} \; v \in S_{k - 1}, \; \text{else “no”}. 
The Reachability Method

Some basic theorems

Non-deterministic Space

Counting the number of reachable nodes (contd.)

Proof.

$\textbf{loop}_2: l := 0; \textbf{for each node } u = 1, 2, \ldots n: \text{if } u \in S(k), \text{ then } l := l + 1.$

How to decide whether $u \in S(k)$?

$\textbf{loop}_3: m := 0; \textbf{reply} = \textbf{false}; \textbf{for each node } v = 1, 2, \ldots n \textbf{ repeat:}$

if $v \in S(k - 1)$ then $m := m + 1$. Further, if $G(v, u)$, then $\textbf{reply} = \textbf{true}.$

if at end, $m < |S(k - 1)|$, then “no”, else return $\textbf{reply}$.

How to check if $v \in S(k - 1)$?

Simple! Start at node $x$ and guess $k - 1$ nodes.

$\textbf{loop}_4: w_0 := x. \textbf{ for } p = 1, 2, \ldots k - 1:$

guess a node $w_p$ and check that $G(w_{p-1}, w_p)$. (If not, return “no”).

if $w_{k-1} = v$, then report $v \in S_{k-1}$, else “no”.

Subramani Complexity Classes
Proof.

loop₂: \( l := 0 \); for each node \( u = 1, 2, \ldots n \): if \( u \in S(k) \), then \( l := l + 1 \).
How to decide whether \( u \in S(k) \)?

loop₃: \( m := 0 \); reply = false; for each node \( v = 1, 2, \ldots n \) repeat:
if \( v \in S(k - 1) \) then \( m := m + 1 \). Further, if \( G(v, u) \), then reply = true.
if at end, \( m < |S(k - 1)| \), then “no”, else return reply.
How to check if \( v \in S(k - 1) \)?

Simple! Start at node \( x \) and guess \( k - 1 \) nodes.

loop₄: \( w_0 := x \). for \( p = 1, 2, \ldots k - 1 \):
guess a node \( w_p \) and check that \( G(w_{p-1}, w_p) \). (If not, return “no”).
if \( w_{k-1} = v \), then report \( v \in S_{k-1} \), else “no”.
Counting the number of reachable nodes (contd.)

Proof.

\textit{loop}_2: \ l := 0; \ \textbf{for} \ \text{each node} \ u = 1, 2, \ldots n: \ \text{if} \ u \in S(k), \ \text{then} \ l := l + 1.

How to decide whether \( u \in S(k) \)?

\textit{loop}_3: \ m := 0; \ \text{reply} = \text{false}; \ \text{for} \ \text{each node} \ v = 1, 2, \ldots n \ \text{repeat}:

\text{if} \ v \in S(k - 1) \ \text{then} \ m := m + 1. \ \text{Further, if} \ G(v, u), \ \text{then} \ \text{reply} = \text{true}.

\text{if} \ \text{at end,} \ m < |S(k - 1)|, \ \text{then “no”, else return \text{reply}}.

How to check if \( v \in S(k - 1) \)?

Simple! Start at node \( x \) and guess \( k - 1 \) nodes.

\textit{loop}_4: \ w_0 := x. \ \textbf{for} \ p = 1, 2, \ldots k - 1:

\text{guess a node} \ w_p \ \text{and check that} \ G(w_{p-1}, w_p). \ \text{(If not, return “no”).}

\text{if} \ w_{k-1} = v, \ \text{then report} \ v \in S_{k-1}, \ \text{else “no”}.
Counting the number of reachable nodes (contd.)

Proof.

\[\text{loop}_2: \ l := 0; \ \textbf{for} \ \text{each node} \ u = 1, 2, \ldots n: \ \text{if} \ u \in S(k), \ \text{then} \ l := l + 1.\]

How to decide whether \( u \in S(k) \)?

\[\text{loop}_3: \ m := 0; \ \text{reply} = \text{false}; \ \text{for each node} \ v = 1, 2, \ldots n \ \text{repeat:}\]

if \( v \in S(k - 1) \) then \( m := m + 1. \) Further, if \( G(v, u) \), then \( \text{reply} = \text{true}. \)

if at end, \( m < |S(k - 1)| \), then “no”, else return \( \text{reply}. \)

How to check if \( v \in S(k - 1) \)?

Simple! Start at node \( x \) and guess \( k - 1 \) nodes.

\[\text{loop}_4: \ w_0 := x. \ \textbf{for} \ p = 1, 2, \ldots k - 1: \]

guess a node \( w_p \) and check that \( G(w_{p-1}, w_p). \) (If not, return “no”).

if \( w_{k-1} = v \), then report \( v \in S_{k-1} \), else “no”.\]
Counting the number of reachable nodes (contd.)

Proof.

\[ \text{loop}_2 : l := 0; \textbf{for} \text{ each node } u = 1, 2, \ldots n : \text{ if } u \in S(k), \text{ then } l := l + 1. \]

How to decide whether \( u \in S(k) \)?

\[ \text{loop}_3 : m := 0; \textit{reply} = \textbf{false}; \textbf{for} \text{ each node } v = 1, 2, \ldots n \text{ repeat:} \]

if \( v \in S(k - 1) \) then \( m := m + 1. \) Further, if \( G(v, u) \), then \( \textit{reply} = \textbf{true}. \)

if at end, \( m < |S(k - 1)|, \) then “no”, else return \( \textit{reply}. \)

How to check if \( v \in S(k - 1) \)?

Simple! Start at node \( x \) and guess \( k - 1 \) nodes.

\[ \text{loop}_4 : w_0 := x. \textbf{for } p = 1, 2, \ldots k - 1: \]

guess a node \( w_p \) and check that \( G(w_{p-1}, w_p). \) (If not, return “no”).

if \( w_{k-1} = v \), then report \( v \in S_{k-1}, \) else “no”.

Subramani Complexity Classes
Corollary

If $f(n) \geq \log n$ is a proper complexity function, then $\text{NSPACE}(f(n)) = \text{coNSPACE}(f(n))$.

Proof.

Let $L \in \text{NSPACE}(f(n))$, i.e., $L$ is decided by a NDTM $M$ that is $f(n)$-space bounded. We construct an NDTM $\tilde{M}$ to decide $\tilde{L}$ as follows: Simply run the algorithm of the Immerman-Szelepséényi theorem on $G(M, x)!$ If $\tilde{M}$ discovers an accepting configuration in any $S(k)$, $k = 0, 1, \ldots, n - 1$, then it halts and rejects. The other possibility is that $|S(n - 1)|$ is computed and no accepting configuration is discovered, in which case $\tilde{M}$ accepts.

We have thus shown that $\text{NSPACE}(f(n)) \subseteq \text{coNSPACE}(f(n))$. The reverse direction can be proved in identical fashion.
Corollary

If \( f(n) \geq \log n \) is a proper complexity function, then \( \text{NSPACE}(f(n)) = \text{coNSPACE}(f(n)) \).

Proof.

Let \( L \in \text{NSPACE}(f(n)) \), i.e., \( L \) is decided by a NDTM \( M \) that is \( f(n) \)-space bounded. We construct an NDTM \( \tilde{M} \) to decide \( L \) as follows: Simply run the algorithm of the Immerman-Szelepsényi theorem on \( G(M, x) \)! If \( \tilde{M} \) discovers an accepting configuration in any \( S(k), k = 0, 1, \ldots, n - 1 \), then it halts and rejects. The other possibility is that \( |S(n - 1)| \) is computed and no accepting configuration is discovered, in which case \( \tilde{M} \) accepts.

We have thus shown that \( \text{NSPACE}(f(n)) \subseteq \text{coNSPACE}(f(n)) \). The reverse direction can be proved in identical fashion.
The Reachability Method

Consequences of counting theorem

Corollary

If \( f(n) \geq \log n \) is a proper complexity function, then \( \text{NSPACE}(f(n)) = \text{coNSPACE}(f(n)) \).

Proof.

Let \( L \in \text{NSPACE}(f(n)) \), i.e., \( L \) is decided by a NDTM \( M \) that is \( f(n) \)-space bounded. We construct an NDTM \( \bar{M} \) to decide \( \bar{L} \) as follows:

Simply run the algorithm of the Immerman-Szelepsényi theorem on \( G(M, x) \). If \( M \) discovers an accepting configuration in any \( S(k) \), \( k = 0, 1, \ldots, n - 1 \), then \( \bar{M} \) halts and rejects. The other possibility is that \( |S(n - 1)| \) is computed and no accepting configuration is discovered, in which case \( \bar{M} \) accepts.

We have thus shown that \( \text{NSPACE}(f(n)) \subseteq \text{coNSPACE}(f(n)) \). The reverse direction can be proved in identical fashion.
Corollary

If \( f(n) \geq \log n \) is a proper complexity function, then \( \text{NSPACE}(f(n)) = \text{coNSPACE}(f(n)) \).

Proof.

Let \( L \in \text{NSPACE}(f(n)) \), i.e., \( L \) is decided by a NDTM \( M \) that is \( f(n) \)-space bounded. We construct an NDTM \( \tilde{M} \) to decide \( \tilde{L} \) as follows: Simply run the algorithm of the Immerman-Szelepscényi theorem on \( G(M, x) \). If \( M \) discovers an accepting configuration in any \( S(k) \), \( k = 0, 1, \ldots, n - 1 \), then \( \tilde{M} \) halts and rejects. The other possibility is that \( |S(n - 1)| \) is computed and no accepting configuration is discovered, in which case \( \tilde{M} \) accepts.

We have thus shown that \( \text{NSPACE}(f(n)) \subseteq \text{coNSPACE}(f(n)) \). The reverse direction can be proved in identical fashion.
Consequences of counting theorem

Corollary

If \( f(n) \geq \log n \) is a proper complexity function, then \( \text{NSPACE}(f(n)) = \text{coNSPACE}(f(n)) \).

Proof.

Let \( L \in \text{NSPACE}(f(n)) \), i.e., \( L \) is decided by a NDTM \( M \) that is \( f(n) \)-space bounded. We construct an NDTM \( \tilde{M} \) to decide \( \bar{L} \) as follows: Simply run the algorithm of the Immerman-Szelepscényi theorem on \( G(M, x) \). If \( \tilde{M} \) discovers an accepting configuration in any \( S(k), k = 0, 1, \ldots, n - 1 \), then it halts and rejects. The other possibility is that \( |S(n - 1)| \) is computed and no accepting configuration is discovered, in which case \( \tilde{M} \) accepts.

We have thus shown that \( \text{NSPACE}(f(n)) \subseteq \text{coNSPACE}(f(n)) \). The reverse direction can be proved in identical fashion.
Corollary

If \( f(n) \geq \log n \) is a proper complexity function, then \( \text{NSPACE}(f(n)) = \text{coNSPACE}(f(n)) \).

Proof.

Let \( L \in \text{NSPACE}(f(n)) \), i.e., \( L \) is decided by a NDTM \( M \) that is \( f(n) \)-space bounded. We construct an NDTM \( \tilde{M} \) to decide \( \tilde{L} \) as follows: Simply run the algorithm of the Immerman-Szelepscényi theorem on \( G(M, x) \). If \( \tilde{M} \) discovers an accepting configuration in any \( S(k), k = 0, 1, \ldots, n - 1 \), then it halts and rejects. The other possibility is that \( |S(n - 1)| \) is computed and no accepting configuration is discovered, in which case \( \tilde{M} \) accepts.

We have thus shown that \( \text{NSPACE}(f(n)) \subseteq \text{coNSPACE}(f(n)) \). The reverse direction can be proved in identical fashion.
Corollary

If \( f(n) \geq \log n \) is a proper complexity function, then \( \text{NSPACE}(f(n)) = \text{coNSPACE}(f(n)) \).

Proof.

Let \( L \in \text{NSPACE}(f(n)) \), i.e., \( L \) is decided by a NDTM \( M \) that is \( f(n) \)-space bounded. We construct an NDTM \( \tilde{M} \) to decide \( \bar{L} \) as follows: Simply run the algorithm of the Immerman-Szelepscényi theorem on \( G(M, x) \)! If \( \tilde{M} \) discovers an accepting configuration in any \( S(k) \), \( k = 0, 1, \ldots, n - 1 \), then it halts and rejects. The other possibility is that \( |S(n - 1)| \) is computed and no accepting configuration is discovered, in which case \( \tilde{M} \) accepts.

We have thus shown that \( \text{NSPACE}(f(n)) \subseteq \text{coNSPACE}(f(n)) \). The reverse direction can be proved in identical fashion.