Relations between Complexity Classes

K. Subramani[1]

[1]Lane Department of Computer Science and Electrical Engineering
West Virginia University

The Reachability Method

Outline

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

Outline

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

Some basic theorems

Theorem

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

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

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

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

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) $\mathbf{SPACE}(f(n)) \subseteq \mathbf{NSPACE}(f(n))$ *and* $\mathbf{TIME}(f(n)) \subseteq \mathbf{NTIME}(f(n))$.

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

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

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) $\textbf{SPACE}(f(n)) \subseteq \textbf{NSPACE}(f(n))$ *and* $\textbf{TIME}(f(n)) \subseteq \textbf{NTIME}(f(n))$.

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

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

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) $\mathbf{SPACE}(f(n)) \subseteq \mathbf{NSPACE}(f(n))$ *and* $\mathbf{TIME}(f(n)) \subseteq \mathbf{NTIME}(f(n))$.

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

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

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) **SPACE**$(f(n)) \subseteq$ **NSPACE**$(f(n))$ *and* **TIME**$(f(n)) \subseteq$ **NTIME**$(f(n))$.

(ii) **NTIME**$(f(n)) \subseteq$ **SPACE**$(f(n))$.

(iii) **NSPACE**$(f(n)) \subseteq$ **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) $\mathbf{SPACE}(f(n)) \subseteq \mathbf{NSPACE}(f(n))$ *and* $\mathbf{TIME}(f(n)) \subseteq \mathbf{NTIME}(f(n))$.

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

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

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) $\mathbf{SPACE}(f(n)) \subseteq \mathbf{NSPACE}(f(n))$ *and* $\mathbf{TIME}(f(n)) \subseteq \mathbf{NTIME}(f(n))$.

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

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

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) $\mathbf{SPACE}(f(n)) \subseteq \mathbf{NSPACE}(f(n))$ *and* $\mathbf{TIME}(f(n)) \subseteq \mathbf{NTIME}(f(n))$.

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

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

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) $\mathbf{SPACE}(f(n)) \subseteq \mathbf{NSPACE}(f(n))$ *and* $\mathbf{TIME}(f(n)) \subseteq \mathbf{NTIME}(f(n))$.

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

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

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) $\mathbf{SPACE}(f(n)) \subseteq \mathbf{NSPACE}(f(n))$ *and* $\mathbf{TIME}(f(n)) \subseteq \mathbf{NTIME}(f(n))$.

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

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

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 (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

$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE}$.

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 ⊆ **NL** ⊆ **P** ⊆ **NP** ⊆ **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

$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE}$.

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**.

Outline

Savitch's Theorem

Theorem

REACHABILITY $\in$ **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:
for all nodes $z \in G$, test whether PATH$(x, z, i-1) \wedge$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

REACHABILITY $\in$ **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:
for all nodes $z \in G$, *test whether* PATH$(x, z, i-1) \wedge$ 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

REACHABILITY $\in$ **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:
for all nodes $z \in G$, *test whether* PATH$(x, z, i-1) \wedge$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

REACHABILITY $\in$ **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:
for all nodes $z \in G$, *test whether* PATH$(x, z, i-1) \wedge$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

REACHABILITY $\in$ **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:
for all nodes $z \in G$, *test whether* PATH$(x, z, i-1) \wedge$ 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

REACHABILITY $\in$ **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:
for all nodes $z \in G$, *test whether* PATH$(x, z, i-1) \wedge$ 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

REACHABILITY $\in$ **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:
for all nodes $z \in G$, *test whether* PATH$(x, z, i-1) \wedge$ 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

REACHABILITY $\in$ **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:
for all nodes $z \in G$, *test whether* PATH$(x, z, i-1) \wedge$ 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

REACHABILITY $\in$ **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:
for all nodes $z \in G$, *test whether* PATH$(x, z, i-1) \wedge$ 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

REACHABILITY $\in$ **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:
for all nodes $z \in G$, *test whether* PATH$(x, z, i-1) \wedge$ 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

REACHABILITY $\in$ **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:
for all nodes $z \in G$, *test whether* PATH$(x, z, i-1) \wedge$ 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

REACHABILITY $\in$ **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:
for all nodes $z \in G$, test whether PATH$(x, z, i-1) \wedge$ 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

REACHABILITY $\in$ **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:
for all nodes $z \in G$*, test whether* PATH$(x, z, i-1) \wedge$ 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

REACHABILITY $\in$ **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:
for all nodes $z \in G$, *test whether* PATH$(x, z, i-1) \wedge$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

REACHABILITY $\in$ **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:
for all nodes $z \in G$, test whether PATH$(x, z, i-1) \wedge$ 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

REACHABILITY $\in$ **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:
for all nodes $z \in G$, *test whether* PATH$(x, z, i-1) \wedge$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

REACHABILITY $\in$ **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:
for all nodes $z \in G$, *test whether* PATH$(x, z, i-1) \wedge$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 (contd.)

Corollary

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

PSPACE = **NPSPACE**.

Savitch's theorem (contd.)

Corollary

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

PSPACE = **NPSPACE**.

Savitch's theorem (contd.)

Corollary

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

PSPACE $=$ **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 $|S(n-1)|$.

$loop_1$: $|S(0)| := 1$; **for** $i = 1, 2, \ldots, n-1$: compute $|S(k)|$ from $|S(k-1)|$. □

Counting the number of reachable nodes

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

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

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

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.

$loop_2$: $l := 0$; **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".

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".

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".

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".

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".

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".

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 $u \in S(k)$?

$loop_3$: $m := 0$; $reply = \textbf{false}$; 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 $reply = \textbf{true}$.
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".

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$_3$: $m := 0$; $reply = \textbf{false}$; 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 $reply = \textbf{true}$.
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".

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$_3$: $m := 0$; $reply = \textbf{false}$; 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 $reply = \textbf{true}$.
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".

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$_3$: $m := 0$; $reply = \textbf{false}$; 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 $reply = \textbf{true}$.
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".

□

Consequences of counting theorem

Corollary

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

Proof.

Let $L \in \mathbf{NSPACE}(f(n))$, 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-Szelepscényi theorem on $G(M, x)$! If $\bar{M}$ 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 $\bar{M}$ accepts.

We have thus shown that $\mathbf{NSPACE}(f(n)) \subseteq \mathbf{coNSPACE}(f(n))$. 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 $\mathbf{NSPACE}(f(n)) = \mathbf{coNSPACE}(f(n))$.

Proof.

Let $L \in \mathbf{NSPACE}(f(n))$, 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-Szelepscényi theorem on $G(M, x)$! If $\bar{M}$ 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 $\bar{M}$ accepts.

We have thus shown that $\mathbf{NSPACE}(f(n)) \subseteq \mathbf{coNSPACE}(f(n))$. 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* $\mathbf{NSPACE}(f(n)) = \mathbf{coNSPACE}(f(n))$.

Proof.

Let $L \in \mathbf{NSPACE}(f(n))$, 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-Szelepscényi theorem on $G(M, x)$! If $\bar{M}$ 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 $\bar{M}$ accepts.

We have thus shown that $\mathbf{NSPACE}(f(n)) \subseteq \mathbf{coNSPACE}(f(n))$. 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 $\mathbf{NSPACE}(f(n)) = \mathbf{coNSPACE}(f(n))$.

Proof.

Let $L \in \mathbf{NSPACE}(f(n))$, 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-Szelepscényi theorem on $G(M, x)$! If $\bar{M}$ 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 $\bar{M}$ accepts.

We have thus shown that $\mathbf{NSPACE}(f(n)) \subseteq \mathbf{coNSPACE}(f(n))$. 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 $\mathbf{NSPACE}(f(n)) = \mathbf{coNSPACE}(f(n))$.

Proof.

Let $L \in \mathbf{NSPACE}(f(n))$, 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-Szelepscényi theorem on $G(M, x)$! If $\bar{M}$ 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 $\bar{M}$ accepts.

We have thus shown that $\mathbf{NSPACE}(f(n)) \subseteq \mathbf{coNSPACE}(f(n))$. 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 **NSPACE**$(f(n))$ = **coNSPACE**$(f(n))$.

Proof.

Let $L \in$ **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-Szelepscényi theorem on $G(M, x)$! If $\bar{M}$ 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 $\bar{M}$ accepts.

We have thus shown that **NSPACE**$(f(n)) \subseteq$ **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 $\textbf{NSPACE}(f(n)) = \textbf{coNSPACE}(f(n))$.

Proof.

Let $L \in \textbf{NSPACE}(f(n))$, 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-Szelepscényi theorem on $G(M, x)$! If $\bar{M}$ 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 $\bar{M}$ accepts.

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

□