**Definition 8.1**

Let $M$ be a deterministic Turing machine, DTM, that halts on all inputs. The space complexity of $M$ is the function $f : \mathbb{N} \rightarrow \mathbb{N}$, where $f(n)$ is the maximum number of tape cells that $M$ scans on any input of length $n$.

If $M$ is a nondeterministic Turing machine, NTM, wherein all branches of its computation halt on all inputs, we define the space complexity of $M$, $f(n)$, to be the maximum number of tape cells that $M$ scans on any branch of its computation for any input of length $n$.

**Example 1 (8.3)**

SAT can be solved with the linear space algorithm $M_1$:

$M_1 = \text{“On input } \langle \Phi \rangle, \text{ where } \Phi \text{ is a Boolean formula:} $

1. For each truth assignment to the variables $x_1, \ldots, x_n$ of $\Phi$:
   (a) Evaluate $\Phi$ on that truth assignment;
   (b) If $\Phi$ evaluates to 1, accept.
2. If $\Phi$ never evaluates to 1, reject.

**Estimation of space complexity**

Let $f : \mathbb{N} \rightarrow \mathbb{N}$ be a function. The space complexity classes, $SPACE(f(n))$ and $NSPACE(f(n))$ are defined by:

- $SPACE(f(n)) = \{ L \mid L \text{ is a language decided by an } O(f(n)) \text{ space DTM} \}$
- $NSPACE(f(n)) = \{ L \mid L \text{ is a language decided by an } O(f(n)) \text{ space NTM} \}$
Example 2

Testing whether a DFA accepts all strings,
\[ \text{ALL}_{\text{DFA}} = \{ \langle A \rangle \mid A \text{ is a DFA and } L(A) = \Sigma^* \} \]
\[ \text{ALL}_{\text{DFA}} = \{ \langle A \rangle \mid A \text{ is a DFA and } L(A) \neq \Sigma^* \} \]
We show that \( \text{ALL}_{\text{DFA}} \in \text{SPACE}(n) \).

Proof idea: construct \( D \), a deterministic linear space algorithm that decide \( \text{ALL}_{\text{DFA}} \) by enumerating all strings of length \( n \). If \( A \) rejects one of them, \( D \) stops with "accept". If \( A \) accepts all, \( D \) will "reject".

Example 3 (8.4)

Testing whether an NFA accepts all strings,
\[ \text{ALL}_{\text{NFA}} = \{ \langle A \rangle \mid A \text{ is an NFA and } L(A) = \Sigma^* \} \]
\[ \text{ALL}_{\text{NFA}} = \{ \langle A \rangle \mid A \text{ is an NFA and } L(A) \neq \Sigma^* \} \]
We show that \( \text{ALL}_{\text{NFA}} \in \text{NSPACE}(n) \).

Proof idea: construct \( N \), a nondeterministic linear space algorithm that decide \( \text{ALL}_{\text{NFA}} \) by guessing a string that is rejected by NFA \( A \) and uses a linear space to keep track of which states the NFA could be in at a particular time. Note, this language is not known to be \( \text{NP} \) or \( \text{co-NP} \).

Space vs NSpace

Recall \( \text{NTIME}(f(n)) \subseteq \text{TIME}(2^{O(f(n))}) \).
Savitch's Theorem For any function \( f : \mathbb{N} \rightarrow \mathbb{N} \), where \( f(n) \geq n \), \( \text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n)) \).
Proof idea: If NTM \( N \) uses \( f(n) \) space, we need to construct a DTM \( M \) which simulates \( N \) and uses \( f^2(n) \) space.

Construction

\( N = \) "On input \( \langle A \rangle \) where \( A \) is an NFA:
1. Place a marker on the start state of \( A \).
2. Repeat \( 2^q \) times, where \( q \) is the number of states of \( A \):
   (a) Nondeterministically select an input symbol and change the positions of all the markers on \( A \)'s states to simulate reading of that symbol.
   (b) If all the marked states are non-final, accept;
3. reject".
**SPACE vs NSPACE**

Recall \( \text{NTIME}(f(n)) \subseteq \text{TIME}(2^{O(f(n))}) \).

**Savitch’s Theorem** For any function \( f: \mathbb{N} \to \mathbb{N} \), where \( f(n) \geq n \), \( \text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n)) \).

**Proof idea:** If NTM \( N \) uses \( f(n) \) space, we need to construct a DTM \( M \) which simulates \( N \) and uses \( f^2(n) \) space.

\( M \) calls the function \( \text{CANYIELD}(c_1, c_2, t) \), which can be computed by a TM, that tests whether a NTM \( N \) can go from the configuration \( c_1 \) of \( N \) to the configuration \( c_2 \) of \( N \) in at most \( t \) steps.

---

**CANYIELD\( (c_1, c_2, t) \)**

\( \text{CANYIELD}(c_1, c_2, t) \) is a TM that tests whether a NTM \( N \) can go from the configuration \( c_1 \) of \( N \) to the configuration \( c_2 \) of \( N \) in at most \( t \) steps.

\( \text{CANYIELD} = \) "On input \( c_1, c_2, t \):

1. If \( t = 1 \) test whether \( c_1 = c_2 \) or whether \( c_1 \) yields \( c_2 \) in one step according to the transition rules of \( N \); accept if either test succeeds, reject is both fail.
2. If \( t > 1 \) then for each configuration \( x \) of \( N \) using space \( f(n) \):
   (a) Run \( \text{CANYIELD}(c_1, x, \lceil t/2 \rceil) \).
   (b) Run \( \text{CANYIELD}(x, c_2, \lceil t/2 \rceil) \).
   (c) If both 2(a) and 2(b) accept then accept.
3. If haven’t yet accepted at 2(c), reject."

---

**Analysis of \( M \)**

- Whenever \( \text{CANYIELD} \) invokes itself recursively it stores the current stage number and the values \( c_1, c_2, t \) on the stack.
- Each level of recursion uses \( O(f(n)) \) additional space.
- Each level of recursion divides the size \( t \) in half. Since initially \( t = 2^d f(n) \), the depth of the recursion is \( O(\log(2^d f(n))) = O(f(n)) \).
- Hence, the total space used is \( O(f^2(n)) \).

---

**DTM \( M \) simulating NTM \( N \)**

Suppose \( N \) is modified that once it enters an accepting state, it erases everything on the tape and moves the tape head to the leftmost position. Let such a final configuration be \( c_2 \).

Let \( d \) be a constant such that \( N \) has no more that \( 2^d f(n) \) configuration.

Then the DTM \( M \) is:

\( M = \) "On input \( w \)

1. Let \( c_s \) be \( q_0 w \), where \( q_0 \) is the start state of \( N \).
2. Output the result of \( \text{CANYIELD}(c_s, c_s, 2^d f(n)) \)"
The class $PSPACE$

$PSPACE$ is the class of languages that are decidable in polynomial space on a DTM, i.e.:

$$PSPACE = \cup_k SPACE(n^k)$$

$$NPSPACE = \cup_k NSPACE(n^k)$$

Let us define $EXPTIME = \cup_k TIME(2^{n^k})$

Lemma: $SPACE(f(n)) \subseteq TIME(2^{O(f(n))})$.

SUMMARY $P \subseteq NP \subseteq PSPACE = NPSPACE \subseteq EXPTIME$

A $PSPACE$-complete language

$TQBF = \{ \langle \Phi \rangle \mid \Phi \text{ is a true fully quantified Boolean formula} \}$

- $\Phi$ is a fully quantified Boolean formula if
  $\Phi = Q_1x_1Q_2x_2\cdots Q_nx_n\Psi$, where $\Psi$ is a Boolean formula in CNF, $x_1, x_2, \ldots, x_n$ are the boolean variables in $\Psi$ ($\Phi$ is said to be in prenex normal form), and $Q_i \in \{\forall, \exists\}$ for $1 \leq i \leq n$.

$TQBF$ is $PSPACE$-complete.

The class $PSPACE$

$PSPACE$ is the class of languages that are decidable in polynomial space on a DTM, i.e.:

$$PSPACE = \cup_k SPACE(n^k)$$

$$NPSPACE = \cup_k NSPACE(n^k)$$

Let us define $EXPTIME = \cup_k TIME(2^{n^k})$

Lemma: $SPACE(f(n)) \subseteq TIME(2^{O(f(n))})$.

$PSPACE$-Completeness

A language $B$ is $PSPACE$-complete if it satisfies two conditions:

1. $B \in PSPACE$
2. Every $A \in PSPACE$ is polynomially time reducible to $B$.

If $B$ satisfies only 2, $B$ is said to be $PSPACE$-hard.
**TQBF is PSPACE-Hard**

Idea: Use $T$ and the construction of $CANYIELD$ to prove that any $PSPACE$ problem reduces to $TQBF$ in polynomial time.

---

**TQBF ∈ PSPACE**

The following polynomial space algorithm decides $TQBF$:

- On input $\langle \Phi \rangle$, a fully quantified Boolean formula:
  1. If $\Phi$ contains no quantifiers, then it is an expression with only constants. So, evaluate $\Phi$ and accept if it is true; otherwise reject.
  2. If $\Phi = \exists x \Psi$, recursively call $T$ on $\Psi$, first with 0 substituted for $x$ and then with 1 substituted for $x$. If either result is "accept", accept; otherwise reject.
  3. If $\Phi = \forall x \Psi$, recursively call $T$ on $\Psi$, first with 0 substituted for $x$ and then with 1 substituted for $x$. If both results are "accept", accept; otherwise reject.

---

**Assignment-tableau correspondence**

The assignment turns on exactly one variable for each cell, using the following constructs:

1. at least one variable that is associated with a cell is on, by: $\bigvee_{s \in C} x_{i,j,s}$
2. no more than one variable is on for each cell, by: $\bigwedge_{s,t \in C, s \neq t} (x_{i,j,s} \lor x_{i,j,t})$

Thus, any satisfying assignment specifies one symbol in each cell by:

$$\Phi_{cell} = \bigwedge_{1 \leq i,j \leq n} (\bigvee_{s \in C} x_{i,j,s} \land \bigwedge_{s \neq t \in C} (x_{i,j,s} \lor x_{i,j,t}))$$

---

**Proof Review: SAT is NP-Hard**

- On input $\langle A, w \rangle$, let $n = |w|$ and $A$ can be recognized by a NTM using $n^k$ time for constant $k$, $f$ will produce $\Phi_w$ in $O(n^k)$.
- Variables of $\Phi_w$: Let $N = (Q, \Sigma, \Gamma, q_0, q_a, q_r)$, $L(N) = A$, and $C = Q \cup \Gamma \cup \{\#\}$. For each $1 \leq i, j \leq n^k$ and $s \in C$ we have a boolean variable $x_{i,j,s}$ in $\Phi_w$.
- Cells: each of the $(n^k)^2$ entries of a tableau is called a cell. $\forall s \in C$ $x_{i,j,s} = 1$ if $cell[i,j] = s$.
- Formula $\Phi_w$: $\Phi_w = \Phi_{cell} \land \Phi_{start} \land \Phi_{move} \land \Phi_{accept}$.
Legal windows

A $2 \times 3$ window of cells is legal if that window does not violate the actions specified by $N$’s transition function. To explain, consider the transitions:

$\delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}$.

Examples of legal windows for this machine are:

<table>
<thead>
<tr>
<th></th>
<th>(a)</th>
<th></th>
<th>(b)</th>
<th></th>
<th>(c)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>$q_1$</td>
<td>$b$</td>
<td>$x$</td>
<td>$q_1$</td>
<td>$b$</td>
<td>$y$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$x$</td>
<td>$c$</td>
<td>$q_2$</td>
<td>$a$</td>
<td>$z$</td>
<td>$q_2$</td>
</tr>
<tr>
<td>$#$</td>
<td>$y$</td>
<td>$x$</td>
<td>$#$</td>
<td>$z$</td>
<td>$x$</td>
<td>$#$</td>
</tr>
</tbody>
</table>

An accepting tableau

is specified by $\Phi_{\text{start}}$, $\Phi_{\text{move}}$, $\Phi_{\text{accept}}$

- $\Phi_{\text{start}}$ ensures that the first row of the tableau is the starting configuration of $N$ on $w$ by the equality:
  
  $\Phi_{\text{start}} = x_1, 1, \# \land x_1, 2, q_0 \land x_1, 3, w_1 \land \ldots \land x_1, n, w_n \land \ldots \land x_1, n+k-1, \# \land x_1, n+k, \#.$

- $\Phi_{\text{accept}}$ guarantees that an accepting configuration occurs in the tableau by placing $q_0$ in one of the cells by:
  
  $\Phi_{\text{accept}} = \bigvee_{1 \leq i, j \leq n^k} x_i, j, q_0.$

- $\Phi_{\text{move}}$ guarantees that each row of the tableau correspond to a configuration that legally follows the preceding row’s configuration according to $N$’s transition rules.

Note

- $\Phi_{c_1, c_2, t}$ encodes the contents of tape cells as in the Cook-Levin theorem. Each configuration has $n^k$ cells and so it is encoded by $O(n^k)$ variables.

  For $t = 1$ the formula $\Phi_{c_1, c_2, 1}$ says that either $c_1 = c_2$ or $c_2$ follows from $c_1$ in a single step of $M$.

  If $t > 1$, construct $\Phi_{c_1, c_2, t}$ recursively:
  
  $\Phi_{c_1, c_2, t} = \exists m_1[\Phi_{c_1, m_1, t/2} \land \Phi_{m_1, c_2, t/2}]$ where $m_1$ is a configuration of $M$. $\exists m_1$ is shorthand for $\exists x_1, x_2, \ldots, x_l = O(n^k)$ and $x_1, \ldots, x_l$ are variables encoding $m_1$.

- To reduce the size of the formula we use both quantifiers, $\forall$ and $\exists$:
  
  $\Phi_{c_1, c_2, t} = \exists m_1 \forall (c_3, c_4) \in \{(c_1, m_1), (m_1, c_2)\}[\Phi_{c_3, c_4, t/2}]$ where $\forall x \in \{g, z, \ldots\} \Phi(x)$ denotes $(\Phi(g) \land P(z) \land \ldots)$.

TQBF is PSPACE-hard

Let $A$ be a language decided by TM $M$ in space $n^k$ for some constant $k$. We give a polynomial time reduction from $A$ to $TQBF$:

The reduction maps machine $M$ and string $w$ to a quantified Boolean formula $\Phi$ that is true iff $M$ accepts $w$:

- The formula is $\Phi_{c_1, c_2, t}$ where $c_1$ and $c_2$ are variables representing two configurations, and $t > 0$.

- If we assign to $c_1$ and $c_2$ actual configurations, $\Phi_{c_1, c_2, t}$ is true iff $M$ can go from $c_1$ to $c_2$ in at most $t$ steps.

- Consider $\Phi_{c_{\text{start}}, x_{\text{accept}}, h}$ where $h = 2^d(n)$ for a constant $d$ chosen so that $M$ has no more than $2^d(n)$ configurations. Assume that $t$ is a power of 2.
A Formula Game

A formula game is a pair \((X, \Psi)\), where \(X\) is a list of boolean variables, \(X = [x_1, x_2, \ldots, x_m]\), and \(\Psi\) is a quantifier-free boolean formula built on \(X\).

The game consists of \(m\) moves: In move \(i\), a player assigns a boolean value to variable \(x_i\).

Two players, \(E\) and \(A\), take turn to make a move.

If \(\Psi\) becomes true after \(m\) moves, \(E\) wins; otherwise, \(A\) wins.

A player is said to have a winning strategy if the player can always win the game by making right moves, no matter how the other player moves.

Example of Formula Game

Suppose \(E\) moves first.

- If \(\Phi = (x_1 \lor \bar{x}_2) \land (x_2 \lor x_3) \land (\bar{x}_2 \lor \bar{x}_3)\) then \(E\) always wins if it selects \(x_1 = 1\), thus \(E\) has the winning strategy.

- If \(\Psi = (x_1 \lor x_2) \land (x_2 \lor x_3) \land (x_2 \lor \bar{x}_3)\) then \(A\) always wins: If \(E\) selects \(x_2 = 1\), then \(A\) selects \(x_1 = 0\); otherwise \(A\) selects \(x_2 = 0\). Thus \(A\) has a winning strategy for this game.

Theorem 8.11

The problem of determining which player has a winning strategy in a formula-game associated with a particular formula is \(PSPACE\)-complete.

Formally:

\[
\text{FORMULA-GAME} = \{ (X, \Phi) \mid \text{Player } E \text{ has a winning strategy in the formula game } (X, \Phi) \}
\]

is \(PSPACE\)-complete.

TQBF vs Formula Game

- Theorem: Given a game \((X, \Psi)\), suppose \(E\) moves first and \(m\) is even. Let \(\Phi = \exists x_1 \forall x_2 \exists x_3 \ldots \forall x_m[\Psi]\). Then \(\Phi\) is true iff \(E\) has a winning strategy for the game \((X, \Psi)\).

- Lemma: For any fully quantified boolean formula \(\Phi\), there exists an equivalent fully quantified boolean formula \(\Phi'\) in prenex normal form whose quantifiers alternate between \(\exists\) and \(\forall\).

- Theorem: For any fully quantified boolean formula \(\Phi\), there exists a formula game \((X, \Psi)\) such that player \(E\) has a winning strategy iff \(\Phi\) is true.
The GG problem

\[ GG = \{ (G, b) \mid \text{Player I has a winning strategy for the generalized geography game played on graph } G \text{ starting at node } b \} \]

Theorem 8.14 \( GG \) is \( \text{PSPACE} \)-complete

Generalized Geography

This is a child game where players take turns naming cities from anywhere in the world. Each city chosen must begin with the same letter that ended the previous city name and no duplication is allowed.

Graph model: A directed graph \( G = (V, E) \) whose nodes are cities of the world and an arrow goes from node \( n_1 \) to node \( n_2 \) if the city labeling \( n_2 \) starts with the letter that ends the name labeling node \( n_1 \).

\( GG \) is \( \text{PSPACE} \)-hard

Theorem: Formula-Game \( \leq_p GG \).
- Let \((X, \Phi)\) be a formula game, where \( \Phi \) is in CNF containing \( m \) clauses. We construct a graph \( G = (V, E) \) and a special node \( b \) such that player \( E \) has a winning strategy in \((X, \Phi)\) for formula game iff player \( E \) has a winning strategy in \((G, b)\) in \( GG \).
- \( V = V_1 \cup V_2 \), where
  - \( V_1 = \{ b_i, x_i, \bar{x}_i, c_{i,j} \mid 1 \leq i \leq k, k = |X| \} \) (assume \( |X| \) is odd), and
  - \( V_2 = \{ c_{j,1}, \ldots, c_{j,k} \mid 1 \leq j \leq m, 1 \leq i \leq k, c_j = (c_{j,1} \lor \cdots \lor c_{j,k}) \} \).
- \( E = E_1 \cup E_2 \cup E_3 \), where
  - \( E_1 \) constructs a chain of “diamonds” among \( V_1 \),
  - \( E_2 \) constructs a tree among \( V_2 \),
  - \( E_3 \) connects \( V_1 \) and \( V_2 \).
- \( b = b_1 \)

\( GG \) is in \( \text{PSPACE} \)

The following algorithm \( M \) decides whether player \( E \) has a winning strategy for \( GG \) game:
\( M = \) “On input \((G, b)\) where \( G \) is a directed graph and \( b \) is a node of \( G \):
  1. If \( b \) has outdegree 0, reject, player \( E \) loses immediately
  2. Remove node \( b \) and all connected arrows to get \( G' \).
  3. For each node \( b_1, b_2, \ldots, b_k \) that \( b \) originally pointed, recursively call \( M \) on \((G', b_1)\).
  4. If one of \((b_i, G')\) returns “reject”, player \( E \) would choose \( b_i \), and \( M \) accepts.
  5. If all of these \((b_i, G')\) return “accept”, player \( A \) has a winning strategy in the original game, so \( M \) returns “reject”.”
**Example**

\[ A = \{0^k1^k \mid k > 0\} \] is a member of \( L \).

- Zig-zag decision algorithm can be modified to use two tapes: a read-only input tape and a read-write working tape.
- The algorithm counts 0-s and 1 in the input and write their binary expressions on the work tape.
- Since only space for two binary numbers is required and since number of binary digits in \( n \) is \( \log n \) it results that \( A \in L \).

**Definition 8.17**

- \( L \) is the class of languages that are decidable in logarithmic space on a DTM with a read-only input tape, i.e., \( L = \text{SPACE}(\log n) \)
- \( NL \) is the class of languages that are decidable in logarithmic space on a NTM with a read-only input tape, i.e., \( L = \text{NSPACE}(\log n) \)

**Note:** Since \( \log n < n \) these languages are also called of sublinear space complexity.

**Log space transducers**

A log space transducer \( M \) is a TM with a read-only input tape, a write-only output tape, and a read/write work tape.

- The work tape may contain \( O(\log n) \) symbols.
- \( M \) computes a function \( f : \Sigma^* \rightarrow \Sigma^* \), where \( f(w) \) is the string remaining on the output tape after \( M \) halts when it starts with \( w \) on the input tape.
- \( f \) is called a log space computable function.

**Example**

\[ \text{PATH} = \{ \langle G, s, t \rangle \mid G \text{ is a directed graph that has a directed path from } s \text{ to } t \} \] is a member of \( NL \).

- Let \( T \) be the NTM recognizing \( \text{PATH} \). \( T \) needs only to remember one node on its work tape. Let that node be \( x \).
- \( T = \) “On input \( \langle G, s, t \rangle \)
  1. Initially, \( x = s \).
  2. If \( x \) has no successors, \( T \) rejects.
  3. \( T \) then nondeterministically chooses a successor \( y \) of \( x \).
  4. If \( y \) is \( t \), \( T \) accepts.
  5. Replace \( x \) by \( y \), go to 2.”
**Configuration**

If $M$ is a TM that has a read-only input tape and a work tape, and $w$ is an input, a configuration of $M$ on $w$ is a setting of the state, the work tape, and the positions of the two tape heads.

**Note:** $|w|$ is, but $w$ itself is not part of the configuration of $M$ on $w$.

If $|w| = n$, $|\Gamma| = g$, $|Q| = h$, and $M$ uses $f(n)$ space, then the different configurations will be $n \times f(n) \times h \times g^{f(n)}$ or $n^{2^{O(f(n))}}$.

**Theorem:** If $A \leq L B$, then $A \leq P B$.

**Log space reducibility**

Language $A$ is log space reducible to language $B$, written $A \leq L B$, if there exists a log space computable function $f$, such that for any $w \in \Sigma^*$, $w \in A$ iff $f(w) \in B$.

**Proof of the Theorem**

**Theorem:** If $A \leq L B$, then $A \leq P B$.

Using Savitch’s Theorem:

For any function $f : \mathbb{N} \rightarrow \mathbb{N}$, where $f(n) \geq n$, $NSPACE(f(n)) \subseteq SPACE(f^2(n))$.

Savitch’s theorem works for $f(n) = O(\log n)$.

**NL-Completeness**

A Language $B$ is NL-complete if:

1. $B \in NL$
2. For every $A \in NL$, $A \leq L B$. 

**NL-Completeness**

A Language $B$ is NL-complete if:

1. $B \in NL$
2. For every $A \in NL$, $A \leq L B$. 

**Proof of the Theorem**

**Theorem:** If $A \leq L B$, then $A \leq P B$.

Using Savitch’s Theorem:

For any function $f : \mathbb{N} \rightarrow \mathbb{N}$, where $f(n) \geq n$, $NSPACE(f(n)) \subseteq SPACE(f^2(n))$.

Savitch’s theorem works for $f(n) = O(\log n)$.
Proof of the Theorem

If $A \leq_L B$ and $B \in L$ then $A \in L$. \textbf{Proof}: Let $M_A$ be the TM recognizing $A$ and $M_B$ be the TM recognizing $B$.

1. $M_A$ computes individual symbols of $f(w)$ as required by $M_B$.
2. In simulation $M_A$ keeps track of where $M_B$’s input head would be on $f(B)$.
3. Every time $M_B$ moves, $M_A$ restarts computation of $f$ on $w$ from the beginning and ignores all the output except for the desired location of $f(w)$. This may be time-inefficient because parts of $f(w)$ could be repeated.
4. However, only a single symbol of $f(w)$ need be stored at any time, in effect trading time for space.

Theorem 8.23

If $A \leq_L B$ and $B \in L$ then $A \in L$.

\textbf{Proof}: for $w \in A$, $f(w) \in B$ but $f(w)$ could require a too large space to fit within log space bound. Hence we need another approach.

Theorem 8.25

PATH is $NL$-complete, where

$PATH = \{ \langle G, s, t \rangle \mid G \text{ is a directed graph that has a directed path from } s \text{ to } t \}$

Corollary

If any $NL$-complete language is in $L$ then $L = NL$. 
Corollary 8.26

\[ NL \subseteq P \]

**Proof:** $PATH \in P$ and any language in $NL$ is polynomially time reducible to $PATH$. 

Summary

\[ L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE = NPSPACE \subseteq EXPTIME \]