Space Complexity
**Polynomial Space Complexity**

**PSPACE**

- $L \in PSPACE \iff$ Exists a Deterministic Turing Machine $M$ that Decides $L$ using at most $O(n^k)$ Positions on the Tape.

**NPSPACE**

- $L \in NPSPACE \iff$ Exists a Non-Deterministic Turing Machine $M$ that Decides $L$ using at most $O(n^k)$ Positions on the Tape for each Path.

**Theorem**

- $PSPACE = NPSPACE$
Examples: Languages in PSPACE

Connected Graph -- $O(n)$

SAT -- $O(n)$

Traveling Salesman -- $O(n)$

Hamiltonian Circuit -- $O(n)$
Proofs: Languages in PSPACE

Connected Graph -- $O(n)$

Proof: Add a Bit to Keep Track of Marking

SAT -- $O(n)$

Proof: Examine All Truth Tables in Lexicographical Order

Traveling Salesman -- $O(n)$

Proof: Depth First Search

Hamiltonian Circuit -- $O(n)$

Proof: Depth First Search
**Space and Time Complexity**

**Deterministic Polynomial**

- Polynomial Time (P) implies Polynomial Space (PSPACE)
- Polynomial Space (PSPACE) does NOT imply Polynomial Time (P)

**Non-Deterministic Polynomial**

- Polynomial Time (NP) implies Polynomial Space (NPSPACE)
- Polynomial Space (NPSPACE) does NOT imply Polynomial Time (NP)
Space and Time Requirements for Decidable Languages

Theorem 1: Space Requirements ≤ Time Requirements

Proof: Obvious

Theorem 2: Time Requirements ≤ $O(c^{Space Requirements})$

Proof: For a Decidable Language the Turing Machine Cannot Loop.

Time Requirements ≤ Maximum Number of Possible Configurations of the Turing Machine on any Input

$$= \underbrace{Number \ of \ States}_{constant} \times \underbrace{(Number \ of \ Tape \ Configurations)}_{(Number \ of \ Tape \ Symbols)^{Space \ Requirements}} \times \underbrace{Number \ of \ Head \ Positions}_{Space \ Requirements}$$

$$\leq O(c^{Space \ Requirements})$$
Turing Machines for Decidable Languages

Properties

• Turing Machine Cannot Loop

• Maximum Number of Possible Configurations of Turing Machine on any Input is:
  \[ \leq O(c^{\text{Space Requirements}}) \]

• *Time Requirements* \( \leq O(c^{\text{Space Requirements}}) \)
Savitch's Theorem

*Exists a Non-Deterministic Turing Machine $M$ that Decides $L$ using at most $O(f(n))$ positions on the Tape*

$\Rightarrow$

*Exists a Deterministic Turing Machine $M^*$ that Decides $L$ using at most $O(f(n)^2)$ positions on the Tape.*

Assumptions

1. $M$ examines the entire tape $\iff f(n) \geq n$.

2. $M$ erases the tape and Halts in unique *accept* state.
Proof: Divide and Conquer:

- Build $\text{CanReach}(T, w, \text{conf}_1, \text{conf}_2, \text{tsteps})$ recursively
  
  If $\text{CanReach}(T, w, \text{conf}_1, \text{conf}_{\text{middle}}, \text{tsteps} / 2)$
  
  and $\text{CanReach}(T, w, \text{conf}_{\text{middle}}, \text{conf}_2, \text{tsteps} / 2)$, then TRUE, else FALSE

Base Cases: $\text{tsteps} = 0, 1$ trivial.

- By Theorem 2 there are only a bounded number of possible configurations.
  
  $\quad \text{maxconf} \leq O(c^{\text{Space Requirements}}) = O(c^{f(n)})$

- Recursively compute $M^* = \text{CanReach}(M, w, c_{\text{start}}, c_{\text{accept}}, t_{\text{maxconfs}})$.
  
  Each recursive call requires 2 Tape descriptions $= O(f(n))$ space
  
  Number of Recursive calls for each Path $= \log(\text{maxconf}) = O(f(n))$

- Deterministic machine $M^*$ uses only $O\left(f(n)^2\right)$ memory.
Corollary 1: \( \text{PSPACE} = \text{NPSPACE} \)

Proof: \( \text{PSPACE} \subseteq \text{NPSPACE} \) Obvious

\( \text{NPSPACE} \subseteq \text{PSPACE} \) Savitch’s Theorem

Corollary 2: \( \text{P} \subseteq \text{NP} \subseteq \text{PSPACE} \)

Proof: \( \text{P} \subseteq \text{NP} \) Obvious

\( \text{NP} \subseteq \text{NPSPACE} = \text{PSPACE} \)
**PSPACE–Complete Languages**

**PSPACE–Hard**

- Every Language $L^*$ in $PSPACE$ is Reducible in Polynomial TIME to $L$

  \[ R : \{ x \in L^* \iff f(x) \in L \} \]

- $L$ not Necessarily in $PSPACE$

**PSPACE–Complete**

- $L$ is in $PSPACE$

- $L$ is $PSPACE$–Hard
Theorem: Suppose that

i. $L_1$ is PSPACE–Complete

ii. $L_2$ is in PSPACE

iii. $L_1$ is Reducible in Polynomial Time to $L_2$
    
    -- Polynomial Time Mapping $R: \{x \in L_1 \iff f(x) \in L_2\}$

Then $L_2$ is PSPACE–Complete.

Proof: $L_1$ PSPACE–Complete $\Rightarrow$ Every Language $L$ in PSPACE is Reducible in Polynomial Time to $L_1$

$\Rightarrow$ Every Language $L$ in PSPACE is Reducible in Polynomial Time to $L_2$

$\Rightarrow$ $L_2$ PSPACE–Complete
Reduction Proof for $PSPACE$–Completeness

$L_1$ $PSPACE$ – Complete

$L_2$ in $PSPACE$ $\implies$ $L_2$ $PSPACE$ – Complete

Polynomial Time Reduction $R : \{x \in L_1 \iff f(x) \in L_2\}$

Need at Least One $PSPACE$–Complete Language to Get Started
Quantified Boolean Expressions

Definition

• Base Case: Boolean Expressions

• Exists: \( \exists P \) \( w \) \( P \) appears in \( w \) as an unbound variable

• All: \( \forall P \) \( w \) \( P \) appears in \( w \) as an unbound variable

Examples

• \( P \lor Q \lor \sim R \)

• \( \exists P (P \lor Q \lor \sim R) \)

• \( \forall R \exists P (P \land Q \land \sim R) \)
Quantified Boolean Formulas

Sentence
• Quantified Boolean Expression where ALL Variables are Bound

Examples
• $P \lor Q \lor \sim R$ NO
• $\forall R \exists P (P \land Q \land \sim R)$ NO
• $\exists Q \forall R \exists P (P \lor Q \lor \sim R)$ YES

QBF
• $\{w \mid w \text{ is a TRUE sentence}\}$
Theorem 1: QBF is in PSPACE.

Proof: Recursively evaluate the expression.

1. \( w \) Consists of All Literals (T or F) -- Simply Evaluate \( w \).

2. \( \forall P \; w \)
   - 2a. Substitute \( P = T \) and evaluate recursively
   - 2b. Substitute \( P = F \) and evaluate recursively
     \( \forall P \; w \) is in QBF if and only if both 2a and 2b evaluate to TRUE.

3. \( \exists P \; w \)
   - 2a. Substitute \( P = T \) and evaluate recursively
   - 2b. Substitute \( P = F \) and evaluate recursively
     \( \forall P \; w \) is in QBF if and only if either 2a and 2b evaluate to TRUE.

All three steps can be done in Linear Space.
Theorem 2: QBF is PSPACE–Complete.

Proof: QBF is in PSPACE (Theorem 1).

Must show QBF is PSPACE–Hard:

- Every Language \( L \) in PSPACE is Reducible in Polynomial Time to QBF

Will define \( R : L \to QBF \)

- \( M \) accepts \( w \) \( \iff \) \( R(w) \) is True

- \( M = \) Deterministic Turing Machine that Decides \( L \)

We follow Ideas in the Proof of Savitch’s Theorem.

Will use Tools from the Proof of the Cook-Levin Theorem.
Divide and Conquer

• Build \( \text{CanReach}(M, \text{conf}_1, \text{conf}_2, \text{tsteps}) \) Recursively.
  (Too Many Configurations -- Rows -- to Describe All \( \text{tsteps} \) Directly.)
  -- \( \text{conf}_1 \) and \( \text{conf}_2 \) are Described by Boolean Expressions
    (see Cook-Levin Theorem for SAT)

• \( \exists \text{conf}_{middle}(\text{CanReach}(M, \text{conf}_1, \text{conf}_{middle}, \text{tsteps} / 2) \)
  \( \land \text{CanReach}(M, \text{conf}_{middle}, \text{conf}_2, \text{tsteps} / 2) \)\)

• \( \exists \text{conf}_{middle}(\forall (c_3, c_4) \in \{(\text{conf}_1, \text{conf}_{middle}), (\text{conf}_{middle}, \text{conf}_2)\}) \)
  \( \text{CanReach}(M, c_3, c_4, \text{tsteps} / 2) \)
Configurations from Cook-Levin Theorem

Literals of $R(w)$

- $Tape(i, j, c) =$ Tape Symbol $c$ Appears in Array Position $(i, j)$
- $State(i, j, q) =$ State Symbol $q$ Appears in Array Position $(i, j)$

Conj #1: Symbols and States

- There is exactly one tape symbol $c$ on the tape in every location.
- There is exactly one state symbol in each row.
Conj #1: Symbols and States

- There is exactly one tape symbol $c$ on the tape in every location.
  -- In each tape location there is only 1 symbol
  \[ T_{i,j} = (\exists c (Tape(i,j,c)) \land (\forall d \neq c (\neg Tape(i,j,d))) \]
  -- In every tape location there is only 1 symbol
  \[ Tapes = \forall_{i,j} T_{i,j} \]

- There is exactly one state symbol in each row.
  -- In each tape location there is only 1 state
  \[ Q_{i,j} = (\exists q (State(i,j,q)) \land (\forall p \neq q (\neg State(i,j,p))) \]
  -- For each row there is exactly 1 column where $Q_{i,j}$ is True
  \[ States = \forall_{rows(i)} (\exists_{column(j)} Q_{i,j} \land (\forall_{columns(k \neq j)} \forall p \neg State(i,k,p))) \]

- Conj #1 = Tapes $\land$ States
Analysis of Space Requirements

• Base Cases: $tsteps = 0, 1$ trivial.

• By Theorem 2 there are only a bounded number of possible configurations.
   -- $maxconf \leq O(c_{Space Requirements}) = O(c_{f(n)})$

• Recursively compute $CanReach(M, c_{start}, c_{accept}, t_{maxconf})$.
   -- Each recursive call requires 3 Tape descriptions $= O(f(n))$ space
   -- Number of Recursive Calls $= \log(maxconf) = O(f(n))$

• $QBF$ expression constructed using only $O(f(n)^2)$ memory and time.
**Sublinear Space Complexity**

*Two Tape Turing Machine*

- Read Only Input Tape
- Read-Write Working Tape

$L$–*SPACE*

- $L^* \in L \iff$ Exists a 2 Tape Deterministic Turing Machine $M$ that Decides $L^*$ using at most $O(\log(n))$ Positions on the Second Tape.

$NL$–*SPACE*

- $L^* \in NL \iff$ Exists a Non-Deterministic Turing Machine $M$ that Decides $L$ using at most $O(\log(n))$ Positions on the Second Tape for each Path.
Observations

Examples
• Balanced Parentheses -- L-Space
• Undirected Path -- NL-Space

Hierarchy
• \( L - SPACE \subseteq NL - SPACE \subseteq PSPACE \)

Open Question
• \( L = NL ? \)

Theorems
• \( L \subseteq P \)
• \( NL \subseteq P \)
Theorem: \( L \subseteq P \)

Proof: Count configurations.

1. Number of possible positions for the read only head is \( n \).

2. Number of possible configurations of the read-write tape is:

   \[
   \text{Number of States} \times (\text{Number of Tape Configurations}) \times \text{Number of Head Positions}
   \]

   \[
   \leq O\left((\text{Space Configurations})^c\right) = O\left(\log(n)c^{\log(n)}\right) = O\left(\log(n)n^{\log(c)}\right)
   \]

   \[
   \{ c^{\log(n)} = n^{\log(c)} \}
   \]

3. Combining 1 and 2 yields a time bound of

   \[
   O\left(n\log(n)n^{\log(c)}\right) \leq O(n^k) \quad k > \log(c) + 2
   \]

   which is polynomial in \( n \).
Additional Results

1. $L \subseteq P$ (Proved)

2. $NL \subseteq P$ (Hard)

3. $L \subseteq NL \subseteq P \subseteq PSPACE$

4. $L = Co - L$ (Obvious)

5. $NL = Co - NL$ (Hard)
Space Complexity Classes

Definition
A function \( s(n) \) is space constructible if \( s(n) \) can be computed in \( O(s(n)) \) space.

\* \( s : \text{unary} \rightarrow \text{binary} \)

Theorem
For any space constructible function \( s(n) \), there is a language \( L_{s(n)}^\text{Hard} \) that is deterministically decidable in \( O(s(n)) \) space, but not in \( o(s(n)) \) space.

Interpretation
More Space \( \Rightarrow \) More Languages