Context Sensitive Languages & Linear Bounded Automaton

By Aravind and Omesh
Guide: Deepak D'Souza

November 25, 2010
Outline

1 CSL
   - Introduction
   - CSL
   - Closure Properties
   - Nonterminal Normal Form
   - Closure Properties
Outline

1. CSL
   - Introduction
   - CSL
   - Closure Properties
   - Nonterminal Normal Form
   - Closure Properties

2. Linear Bounded Automata
   - Introduction
   - LBA eg
   - LBA Decidability
   - LBA and Emptiness
   - LBA and Reachability
   - LBA Complement
   - LBA Union and Intersection
**Introduction**

*Introduce Context Sensitive Languages and Linear Bound Automaton*
Introduction

*Introduce Context Sensitive Languages and Linear Bound Automaton*

- $G = (V, \Sigma, R, S)$
Introduction

Introduce Context Sensitive Languages and Linear Bound Automaton

- \( G = (V, \Sigma, R, S) \)
- \( V \) is set of variables
Introduction

Introduce Context Sensitive Languages and Linear Bound Automaton

- \( G = (V, \Sigma, R, S) \)
- \( V \) is set of variables
- \( \Sigma \) is set of terminals
Introduction

*Introduce Context Sensitive Languages and Linear Bound Automaton*

- $G = (V, \Sigma, R, S)$
- $V$ is set of variables
- $\Sigma$ is set of terminals
- $S$ is start symbol
Introduction

*Introduce Context Sensitive Languages and Linear Bound Automaton*

- \( G = (V, \Sigma, R, S) \)
- \( V \) is set of variables
- \( \Sigma \) is set of terminals
- \( S \) is start symbol
- \( R \) are rules of the form \( \alpha A \beta \rightarrow \alpha \gamma \beta \)
  where \( A \in V, \alpha, \beta \in (V \cup \Sigma)^* \), and \( \gamma \in (V \cup \Sigma)^+ \)
  plus the rule \( S \Rightarrow \epsilon \) if \( S \) is not on right side of any rule.
Context Sensitive Languages eg

- CSL for \( \{a^n b^n c^n \mid n > 0\} \)
Context Sensitive Languages eg

- CSL for \( \{a^n b^n c^n \mid n > 0\} \)

\[ S \rightarrow abc | aAbc \]
Context Sensitive Languages eg

- CSL for \( \{a^n b^n c^n \mid n > 0 \} \)

\[
S \rightarrow abc | aA bc \\
Ab \rightarrow bA
\]
Context Sensitive Languages eg

- CSL for \( \{a^n b^n c^n \mid n > 0\} \)

\[
S \rightarrow abc | aAbc \\
Ab \rightarrow bA \\
Ac \rightarrow BbCc
\]
Context Sensitive Languages eg

- CSL for \( \{a^n b^n c^n \mid n > 0\} \)

\[
S \rightarrow \text{abc|aAbc}
\]
\[
Ab \rightarrow bA
\]
\[
Ac \rightarrow Bb\text{cc}
\]
\[
bB \rightarrow Bb
\]
Context Sensitive Languages eg

- CSL for \( \{a^n b^n c^n \mid n > 0\} \)

\[
S \rightarrow abc | aA bc \\
Ab \rightarrow bA \\
Ac \rightarrow Bbcc \\
bB \rightarrow Bb \\
aB \rightarrow aa | aaA
\]
Context Sensitive Languages eg

- CSL for \{a^3b^3c^3 \mid n > 0\}
Context Sensitive Languages eg

- CSL for \( \{a^3b^3c^3 \mid n > 0\} \)

\[
S \rightarrow aAbc
\]
Context Sensitive Languages eg

- CSL for \( \{a^3b^3c^3 \mid n > 0\} \)

\[
S \rightarrow aAbc \rightarrow abAc
\]
Context Sensitive Languages eg

- CSL for \( \{a^3b^3c^3 \mid n > 0\} \)

S → aAbc → abAc → abBbcc
Context Sensitive Languages eg

- CSL for \( \{a^3b^3c^3 \mid n > 0\} \)

\[
S \rightarrow aAbc \rightarrow abAc \rightarrow abBbCc \rightarrow aBbbCc
\]
Context Sensitive Languages eg

- CSL for \( \{a^3b^3c^3 \mid n > 0\} \)

\[
S \rightarrow aAbc \rightarrow abAc \rightarrow abBbCc \rightarrow aBbbCc \\
\rightarrow aaAbbCc
\]
Context Sensitive Languages eg

- CSL for $\{a^3b^3c^3 \mid n > 0\}$

  
  \[
  S \rightarrow aAbc \rightarrow abAc \rightarrow abBbcc \rightarrow aBbccc \\
  \rightarrow aaAbbccc \rightarrow aabAbcc
  \]
Context Sensitive Languages eg

- CSL for \( \{a^3b^3c^3 \mid n > 0\} \)

\[
S \rightarrow aAbc \rightarrow abAc \rightarrow abBbCc \rightarrow aBbbCc \\
\rightarrow aaAbbCc \rightarrow aabAbcc \rightarrow aabbAcc
\]
Context Sensitive Languages eg

- CSL for \( \{a^3b^3c^3 \mid n > 0\} \)

\[
S \rightarrow aAbc \rightarrow abAc \rightarrow abBbcc \rightarrow aBbbcc \\
\rightarrow aaAbbccc \rightarrow aabAbcc \rightarrow aabbAcc \rightarrow aabbBbccc
\]
Context Sensitive Languages eg

- CSL for \( \{a^3b^3c^3 \mid n > 0\} \)

\[
S \rightarrow aAbc \rightarrow abAc \rightarrow abBbcc \rightarrow aBbbcc \\
\quad \rightarrow aaAbbcc \rightarrow aabAbcc \rightarrow aabbAcc \rightarrow aabbBbccc \\
\quad \rightarrow aabBbbccc
\]
Context Sensitive Languages eg

- CSL for \( \{a^3b^3c^3 \mid n > 0\} \)

\[
S \rightarrow aAbc \rightarrow abAc \rightarrow abBbcc \rightarrow aBbbcc \\
\rightarrow aaAbbcc \rightarrow aabAbcc \rightarrow aabbAcc \rightarrow aabbBbccc \\
\rightarrow aabBbbccc \rightarrow aaBbbbcc
\]
Context Sensitive Languages eg

- CSL for \{a^3b^3c^3 \mid n > 0\}

\[ S \rightarrow aAbc \rightarrow abAc \rightarrow abBbCc \rightarrow aBbbCc \]
\[ \rightarrow aaAbbCc \rightarrow aabAbCc \rightarrow aabbAcc \rightarrow aabbBbCcCc \]
\[ \rightarrow aabBbbCcCc \rightarrow aaBbbbCcCc \rightarrow aaabbbCcCc \]
CFL $\subseteq$ CSL
CFL ⊆ CSL
CFL $\subseteq$ CSL

- CFG $\rightarrow$ CNF
CFL ⊆ CSL

- CFG → CNF → CSL
CSL, G generates w is decidable
CSL, G generates $w$ is decidable

- Non deterministically apply the transitions starting from start state
CSL, G generates w is decidable

- Non deterministically apply the transitions starting from start state
- G will generate w
CSL, G generates $w$ is decidable

- Non deterministically apply the transitions starting from start state
- G will generate $w$
- G will generate string to which no rule can be applied
CSL, $G$ generates $w$ is decidable

- Non deterministically apply the transitions starting from start state
- $G$ will generate $w$
- $G$ will generate string to which no rule can be applied
- $G$ will keep generating strings of same length
CSL, G generates w is decidable

- Non deterministically apply the transitions starting from start state
- G will generate w
- G will generate string to which no rule can be applied
- G will keep generating strings of same length
- Since there is a finite no of strings of given length, G must generate someone twice
CSL, G generates w is decidable

- Non deterministically apply the transitions starting from start state
- G will generate w
- G will generate string to which no rule can be applied
- G will keep generating strings of same length
- Since there is a finite no of strings of given length, G must generate someone twice
- G will generate S, that is longer than w
CSL, G generates w is decidable

- Non deterministically apply the transitions starting from start state
- G will generate w
- G will generate string to which no rule can be applied
- G will keep generating strings of same length
- Since there is a finite no of strings of given length, G must generate someone twice
- G will generate S, that is longer than w
- The path can be terminated
There are decidable languages which are not CSL.
There are decidable languages which are not CSL

- Diagonalization argument
There are decidable languages which are not CSL

- Diagonalization argument
- Encode the non terminal alphabet as $\times d_1, d_2, \ldots, d_n$ where $d_i$ belongs to $\{0,1\}$
There are decidable languages which are not CSL

- Diagonalization argument
- Encode the non terminal alphabet as $\times d_1, d_2, \ldots, d_n$ where $d_i$ belongs to $\{0,1\}$
- Terminal alphabets $\{a,b\}$
There are decidable languages which are not CSL

- Diagonalization argument
- Encode the non terminal alphabet as $\times d_1, d_2, \ldots, d_n$ where $d_i$ belongs to $\{0,1\}$
- Terminal alphabets $\{a,b\}$
- Encode each rule
There are decidable languages which are not CSL

- Diagonalization argument
- Encode the non terminal alphabet as $d_1, d_2, \ldots, d_n$ where $d_i$ belongs to $\{0,1\}$
- Terminal alphabets $\{a,b\}$
- Encode each rule
- Encode $G$ by concatenating together its rules separated by ‘;’s
There are decidable languages which are not CSL
Closure Properties

Union
Closure Properties

Union

- CSL are closed under Union.
Closure Properties

**Union**

- CSL are closed under Union.
- **Proof: By Construction**
Consider Context Sensitive Grammar $G_1 = (V_1, \Sigma_1, R_1, S_1)$ and $G_2 = (V_2, \Sigma_2, R_2, S_2)$. Rename non-terminals of $G_1$ and $G_2$, so that two sets are disjoint and neither one contains $S$. Build new Grammar $G = (V_1 \cup V_2 \cup S, \Sigma_1 \cup \Sigma_2, R_1 \cup R_2 \cup \{S \Rightarrow S_1 \cup S_2\}, S)$.
Closure Properties

**Union**

- Consider Context Sensitive Grammar $G_1 = (V_1, \Sigma_1, R_1, S_1)$ and $G_2 = (V_2, \Sigma_2, R_2, S_2)$
Closure Properties

**Union**

- Consider Context Sensitive Grammar $G_1 = (V_1, \Sigma_1, R_1, S_1)$ and $G_2 = (V_2, \Sigma_2, R_2, S_2)$
- Rename Non-Terminals of $G_1$ and $G_2$, so that two sets are disjoint and neither one contains $S$. 
Consider Context Sensitive Grammar $G_1 = (V_1, \Sigma_1, R_1, S_1)$ and $G_2 = (V_2, \Sigma_2, R_2, S_2)$.

Rename Non-Terminals of $G_1$ and $G_2$, so that two sets are disjoint and neither one contains $S$.

Build new Grammar $G = (V_1 \cup V_2 \cup S, \Sigma_1 \cup \Sigma_1, R_1 \cup R_2 \cup \{S \Rightarrow S_1 \cup S_2\}, S)$.
Consider Context Sensitive Grammar $G_1 = (V_1, \Sigma_1, R_1, S_1)$ and $G_2 = (V_2, \Sigma_2, R_2, S_2)$.

Rename Non-Terminals of $G_1$ and $G_2$, so that two sets are disjoint and neither one contains $S$.

Build new Grammar $G = (V_1 \cup V_2 \cup S, \Sigma_1 \cup \Sigma_2, R_1 \cup R_2 \cup \{S \Rightarrow S_1S_2\}, S)$

*By Aravind and Omesh Guide: Deepak D’Souza*
Closure Properties

**Concatenation**

- Consider Context Sensitive Grammar $G_1 = (V_1, \Sigma_1, R_1, S_1)$ and $G_2 = (V_2, \Sigma_2, R_2, S_2)$
Closure Properties

Concatenation

Consider Context Sensitive Grammar $G_1 = (V_1, \Sigma_1, R_1, S_1)$ and $G_2 = (V_2, \Sigma_2, R_2, S_2)$

Rename Non-Terminals of $G_1$ and $G_2$, so that two sets are disjoint and neither one contains $S$. 
Closure Properties

**Concatenation**

- Consider Context Sensitive Grammar $G_1 = (V_1, \Sigma_1, R_1, S_1)$ and $G_2 = (V_2, \Sigma_2, R_2, S_2)$

- Rename Non-Terminals of $G_1$ and $G_2$, so that two sets are disjoint and neither one contains $S$.

- Build new Grammar $G = (V_1 \cup V_2 \cup S, \Sigma_1 \cup \Sigma_1, R_1 \cup R_2 \cup \{ S \Rightarrow S_1 S_2 \}, S)$
Closure Properties

What Goes Wrong?

```
S
 / \  
S₁   S₂
  / \  /  
a  A  a  a  b
```
Closure Properties

What Goes Wrong?

Two Subtrees may interact.
A Context Sensitive Grammar $G = (V, \Sigma, R, S)$ is in nonterminal normal form iff all rules in $R$ one of the following two forms:

1. $\alpha \rightarrow C$, where $\alpha \in (V)$ and $C \in \Sigma$
2. $\alpha \rightarrow \beta$, where both $\alpha, \beta \in (V)$

By Aravind and Omesh Guide: Deepak D'Souza

Context Sensitive Languages & Linear Bounded Automaton
Nonterminal Normal Form

A Context Sensitive Grammar $G = (V, \Sigma, R, S)$ is in nonterminal normal form iff all rules in $R$ one of the following two forms:
Nonterminal Normal Form

A Context Sensitive Grammar $G = (V, \Sigma, R, S)$ is in nonterminal normal form iff all rules in $R$ one of the following two forms:

- $\alpha \Rightarrow C$, where $\alpha \in (V)$ and $C \in \Sigma$
A Context Sensitive Grammar $G = (V, \Sigma, R, S)$ is in nonterminal normal form iff all rules in $R$ one of the following two forms:

1. $\alpha \Rightarrow C$, where $\alpha \in (V)$ and $C \in \Sigma$
2. $\alpha \Rightarrow \beta$, where both $\alpha, \beta \in (V)^+$

Nonterminal Normal Form
Closure Properties

Nonterminal Normal Form

Given a Context Sensitive Grammar $G = (V, \Sigma, R, S)$ there exists an equivalent non-terminal normal form grammar $G'$ such that $L(G) = L(G')$.

Proof by Construction:
For each terminal symbol $c$ in $\Sigma$, create a new Non-Terminal symbol $T_c$ and add to $R$, the rule $T_c \Rightarrow c$.
Nonterminal Normal Form

Given a Context Sensitive Grammar $G = (V, \Sigma, R, S)$ there exists an equivalent non-terminal normal form grammar $G'$ such that $L(G) = L(G')$.
Nonterminal Normal Form

- Given a Context Sensitive Grammar \( G = (V, \Sigma, R, S) \) there exists an equivalent non-terminal normal form grammar \( G' \) such that \( L(G) = L(G') \)

- Proof by Construction:
Nonterminal Normal Form

- Given a Context Sensitive Grammar $G = (V, \Sigma, R, S)$ there exists an equivalent non-terminal normal form grammar $G'$ such that $L(G) = L(G')$

- Proof by Construction:

- For each terminal symbol $c$ in $\Sigma$, create a new Non-Terminal symbol $T_c$ and add to $R$, the rule $T_c \Rightarrow c$
Closure Properties

Concatenation Revisited
Closure Properties

*Concatenation Revisited*

- Convert Both $G_1$ and $G_2$ to Nonterminal Normal Form.
Closure Properties

**Concatenation Revisited**

- Convert Both $G_1$ and $G_2$ to Nonterminal Normal Form.
- Rename the nonterminals of $G_1$ and $G_2$ so that the two sets are disjoint and neither includes symbol $S$. 
Concatenation Revisited

- Convert Both $G_1$ and $G_2$ to Nonterminal Normal Form.
- Rename the nonterminals of $G_1$ and $G_2$ so that the two sets are disjoint and neither includes symbol $S$.
- Build new Grammar $G = (V_1 \cup V_2 \cup S, \Sigma_1 \cup \Sigma_1, \{S \Rightarrow S_1 S_2\}, S)$
Closure Properties

*Kleene Star*
Closure Properties

*Kleene Star*

- Given a Context Sensitive Grammar $G_1 = (V_1, \Sigma_1, R_1, S_1)$
Closure Properties

**Kleene Star**

- Given a Context Sensitive Grammar $G_1 = (V_1, \Sigma_1, R_1, S_1)$
- Construct a Context Sensitive Grammar $G = (V, \Sigma, R, S)$ such that $L_1 = L(G_1)^*$
Closure Properties

Kleene Star

- Given a Context Sensitive Grammar $G_1 = (V_1, \Sigma_1, R_1, S_1)$
- Construct a Context Sensitive Grammar $G = (V, \Sigma, R, S)$ such that $L_1 \equiv L(G_1)^*$
- Cannot use $S \Rightarrow \epsilon \mid S_1S$
Closure Properties

*Kleene Star*

- Given a Context Sensitive Grammar $G_1 = (V_1, \Sigma_1, R_1, S_1)$
- Construct a Context Sensitive Grammar $G = (V, \Sigma, R, S)$ such that $L_1 = L(G_1)^*$
- Cannot use $S \Rightarrow \epsilon \mid S_1S$
- Add two Nonterminals $S$ and $T$ and the rules $S \Rightarrow \epsilon \mid T$
  $T \Rightarrow T S_1 \mid S_1$
Closure Properties

*Kleene Star*

```
S
 /    |
T     T
/  |
S1   S1
/ |
S1 W
/ a
A
A
b
```
Closure Properties

*Kleene Star*

- Problem with previous construction.

---

By Aravind and Omesh Guide: Deepak D'Souza

Context Sensitive Languages & Linear Bounded Automaton
Closure Properties

*Kleene Star*
Closure Properties

Kleene Star

Solution
Closure Properties

*Kleene Star*

- Solution
- Convert $G_1$ to Nonterminal normal form.
Closure Properties

**Kleene Star**

- Solution
- Convert $G_1$ to Nonterminal normal form.
- Rename Nonterminals so they donot include $S,T,T'$ and $S_2$
Closure Properties

*Kleene Star*

- **Solution**
- Convert $G_1$ to Nonterminal normal form.
- Rename Nonterminals so they don't include $S,T,T'$ and $S_2$
- Create new Nonterminal $S_2$ and create copies (with different names) of all the Nonterminals and the rules in $G_1$ so that $L(S_2) = L(S_1)$
Closure Properties

**Kleene Star**

- Solution
- Convert $G_1$ to Nonterminal normal form.
- Rename Nonterminals so they donot include $S, T, T', S_2$.
- Create new Nonterminal $S_2$ and create copies (with different names) of all the Nonterminals and the rules in $G_1$ so that $L(S_2) = L(S_1)$.
- Add the Rules $S \Rightarrow \epsilon$, $S \Rightarrow T$, $T \Rightarrow T' S_1$, $T \Rightarrow S_1$, $T' \Rightarrow TS_2$, $T' \Rightarrow S_2$. 

By Aravind and Omesh Guide: Deepak D'Souza

Context Sensitive Languages & Linear Bounded Automaton
Closure Properties

*Kleene Star*
Intersection and Complement
Intersection and Complement

- Intersection simulate two LBA’s on one.
Intersection and Complement

- Intersection simulate two LBA’s on one.
- Complement Miel Immerman
A Turing machine that has the length of its tape limited to the length of the input string is called a linear-bounded automaton (LBA).
Introduction

- A Turing machine that has the length of its tape limited to the length of the input string is called a linear-bounded automaton (LBA).
- A linear-bounded automaton (LBA) is an 7-tuple.
Introduction

- \( M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}) \)
Introduction

- $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$
- $Q$ is the finite set of states
Introduction

- $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$
- $Q$ is the finite set of states
- $\Sigma$ is the input alphabet
Introduction

- $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$
- $Q$ is the finite set of states
- $\Sigma$ is the input alphabet
- $\Gamma$ is the tape alphabet ($\Sigma \subseteq \Gamma$)
Introduction

- $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$
- $Q$ is the finite set of states
- $\Sigma$ is the input alphabet
- $\Gamma$ is the tape alphabet ($\Sigma \subseteq \Gamma$)
- $<$ is the left endmarker
Introduction

- $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$
- $Q$ is the finite set of states
- $\Sigma$ is the input alphabet
- $\Gamma$ is the tape alphabet ($\Sigma \subseteq \Gamma$)
- $<$ is the left endmarker
- $>$ is the right endmarker
Introduction

- \( M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}) \)
- \( Q \) is the finite set of states
- \( \Sigma \) is the input alphabet
- \( \Gamma \) is the tape alphabet (\( \Sigma \subseteq \Gamma \))
- \( < \) is the left endmarker
- \( > \) is the right endmarker
- \( q_0 \) is the initial state
Introduction

- $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$
- $Q$ is the finite set of states
- $\Sigma$ is the input alphabet
- $\Gamma$ is the tape alphabet ($\Sigma \subseteq \Gamma$)
- $<$ is the left endmarker
- $>$ is the right endmarker
- $q_0$ is the initial state
- $q_{accept}$ is the set of accept state
Introduction

- \( M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject}) \)
- \( Q \) is the finite set of states
- \( \Sigma \) is the input alphabet
- \( \Gamma \) is the tape alphabet (\( \Sigma \subseteq \Gamma \))
- \(<\) is the left endmarker
- \(>\) is the right endmarker
- \( q_0 \) is the initial state
- \( q_{accept} \) is the set of accept state
- \( q_{reject} \) is the set of reject state

By Aravind and Omesh Guide: Deepak D'Souza  
Context Sensitive Languages & Linear Bounded Automaton
LBA eg

\{ a^n b^n c^n \mid n > 0 \}
LBA eg

- \{a^n b^n c^n \mid n > 0\}
- Q = \{s, t, u, v, w\}
LBA eg

- $\{a^n b^n c^n \mid n > 0\}$
- $Q = \{s, t, u, v, w\}$
- $\Sigma = a, b, c$
LBA eg

- $\{a^n b^n c^n \mid n > 0\}$
- $Q = \{s,t,u,v,w\}$
- $\Sigma = a,b,c$
- $\Gamma = a,b,c,x$
LBA eg

- \( \{ a^n b^n c^n \mid n > 0 \} \)
- \( Q = \{ s, t, u, v, w \} \)
- \( \Sigma = a, b, c \)
- \( \Gamma = a, b, c, x \)
- \( q_0 = s \)
LBA eg

\[ \delta = \]

[Outline]

- CSL
- Linear Bounded Automata

[Introduction]

- LBA eg
- LBA Decidability
- LBA and Emptiness
- LBA and Reachability
- LBA Complement
- LBA Union and Intersection
LBA eg

\[ \delta = \{
(\langle s, \langle \rangle, \langle t, \langle, R \rangle \rangle, (\langle t, \rangle, \langle t, \rangle, L)), \\
(\langle t, x \rangle, \langle t, x, R \rangle), (\langle t, a \rangle, \langle u, x, R \rangle), \\
(\langle u, a \rangle, \langle u, a, R \rangle), (\langle u, x \rangle, \langle u, x, R \rangle), \\
(\langle u, b \rangle, \langle v, x, R \rangle), (\langle v, b \rangle, \langle v, b, R \rangle), \\
(\langle v, x \rangle, \langle v, x, R \rangle), (\langle v, c \rangle, \langle w, x, L \rangle), \\
(\langle w, c \rangle, \langle w, c, L \rangle), (\langle w, b \rangle, \langle w, b, L \rangle), \\
(\langle w, a \rangle, \langle w, a, L \rangle), (\langle w, x \rangle, \langle w, x, L \rangle), \\
(\langle w, \langle \rangle, \langle t, \langle, R \rangle \rangle)
\} \]
LBA Decidability

A_{LBA} = \{ \langle M,w \rangle | M \text{ is an LBA and } M \text{ accepts } w \}\n
Unlike \( A_{TM} \), \( A_{LBA} \) is decidable.

Proof:
The ID of an LBA (like a TM) consists of the current tape contents (\( w_i \)), the current state (\( q \)), and the current head position. For a Turing machine, there are infinitely many IDs. However, for an LBA, there are a finite number. Precisely, there are \( n \times |Q| \times |\Gamma| \) possible IDs where \( n \) is the length of the input string. \( |\Gamma| \) is the number of possible tape strings. \( |Q| \) is the number of possible states. And \( n \) is the number of head positions.
LBA Decidability

\[ A_{LBA} = \{ <M,w> \mid M \text{ is an LBA and } M \text{ accepts } w \} \]
LBA Decidability

- $A_{LBA} = \{ <M, w> | M \text{ is an LBA and } M \text{ accepts } w \}$
- Unlike $A_{TM}$, $A_{LBA}$ is decidable
LBA Decidability

- \( A_{LBA} = \{ <M,w> | M \text{ is an LBA and } M \text{ accepts } w \} \)
- Unlike \( A_{TM} \), \( A_{LBA} \) is decidable
- Proof:
LBA Decidability

- \( A_{LBA} = \{ <M,w> | M \) is an LBA and \( M \) accepts \( w \} \)
- Unlike \( A_{TM} \), \( A_{LBA} \) is decidable
- Proof:

The ID of an LBA (like a TM) consists of the current tape contents (\( wi \)), the current state (q), and the current head position. For a turing machine, there are infinitely many IDs. However, for an LBA, there are a finite number. Precisely, there are \( n \times |Q| \times |\Gamma|^n \) possible IDs where \( n \) is the length of the input string. \( |\Gamma|^n \) is the number of possible tape strings. \( |Q| \) is the number of states. And \( n \) is the number of head positions.
Computation on LBA can be defined by ID's

\[ ID_0 \vdash ID_1 \vdash \ldots \vdash ID_k, \text{where } ID_0 \text{ is an initial configuration} \]

If an ID appears twice, then the machine is in a loop.

On input \( \{ M, w \} \), where \( M \) is an LBA and \( w \) is an input word,
1. Simulate machine \( M \) for at most \( n \times |Q| \times |\Gamma| \) steps of computation.
2. If \( M \) accepted, accept. If \( M \) rejected, reject. Otherwise, \( M \) must be in a loop; reject.

By Aravind and Omesh Guide: Deepak D'Souza
LBA Decidability

- Computation on LBA can be defined by ID’s
LBA Decidability

- Computation on LBA can be defined by ID’s
- ID0 ⊨ ID1 ⊨ ... ⊨ IDk, where ID0 is an initial configuration
LBA Decidability

- Computation on LBA can be defined by ID’s
- ID0 ⊢ ID1 ⊢ ... ⊢ IDk, where ID0 is an initial configuration
- If an ID appears twice, then the machine is in a loop.
LBA Decidability

- Computation on LBA can be defined by ID’s
- \( ID_0 \vdash ID_1 \vdash \ldots \vdash ID_k \), where \( ID_0 \) is an initial configuration
- If an ID appears twice, then the machine is in a loop.
- On input \( \{M, w\} \), where \( M \) is an LBA and \( w \) is an input word,
LBA Decidability

- Computation on LBA can be defined by ID’s
- $ID_0 \vdash ID_1 \vdash ... \vdash ID_k$, where $ID_0$ is an initial configuration
- If an ID appears twice, then the machine is in a loop.
- On input $\{M,w\}$, where $M$ is an LBA and $w$ is an input word,
  1. Simulate machine $M$ for at most $n \times |Q| \times |\Gamma|^n$ steps of computation.
LBA Decidability

- Computation on LBA can be defined by ID’s
- $\text{ID}_0 \vdash \text{ID}_1 \vdash \ldots \vdash \text{ID}_k$, where $\text{ID}_0$ is an initial configuration
- If an ID appears twice, then the machine is in a loop.
- On input $\{M, w\}$, where $M$ is an LBA and $w$ is an input word,
  1. Simulate machine $M$ for at most $n \times |Q| \times |\Gamma|^n$ steps of computation.
  2. If $M$ accepted, accept. If $M$ rejected, reject. Otherwise, $M$ must be in a loop; reject.
LBA Decidability

Corollary

The membership problems for sets accepted by linear bounded automata are solvable.

The sets accepted by linear bounded automata are all recursive.
LBA Decidability

- Corollary
LBA Decidability

- Corollary
- The membership problems for sets accepted by linear bounded automata are solvable
LBA Decidability

- **Corollary**
  - The membership problems for sets accepted by linear bounded automata are solvable
  - The sets accepted by linear bounded automata are all recursive.
LBA Emptiness
LBA Emptiness

\[ E_{LBA} = \{ \langle M \rangle \mid M \text{ is an LBA and } L(M) = \emptyset \} \text{ is undecidable} \]
LBA Emptiness

\[ E_{LBA} = \{ < M > \mid M \text{ is an LBA and } L(M) = \emptyset \} \] is undecidable

Proof:
Lemma: For any nondeterministic linear bounded automaton there is another which can locate and examine \( m \) configurations reachable (by the first LBA) from some input if there are at least \( m \) reachable configurations.
Reachability

- **Lemma:** For any nondeterministic linear bounded automaton there is another which can locate and examine $m$ configurations reachable (by the first LBA) from some input if there are at least $m$ reachable configurations.

- **Proof:** We have an NLBA and an integer $m$. In addition we know that there are at least $m$ configurations reachable from a certain input. Our task is to find them.
Reachability algorithm

\[ x = 0 \]
\[ \text{for } i = 1 \text{ to } k \times n \times s^n \]
\[ \text{generate } C_i; \]
\[ \text{guess a path from } C_0 \text{ to } C_i; \]
\[ \text{verify that it is a proper path} \]
\[ \text{if } C_i \text{ is reachable then } x = x + 1 \]
\[ \text{verify that } x \geq m \text{ (otherwise reject)} \]
Reachability algorithm

\[ x = 0 \]
\[ \text{for } i = 1 \text{ to } k \times n \times s^n \]
\[ \text{generate } C_i \]
\[ \text{guess a path from } C_0 \text{ to } C_i \]
\[ \text{verify that it is a proper path} \]
\[ \text{if } C_i \text{ is reachable then } x = x + 1 \]
\[ \text{verify that } x \geq m \text{ (otherwise reject)} \]

- We looked at all possible configurations and counted those which were reachable from the starting configuration.
Lemma: For any nondeterministic linear bounded automaton there is another which can compute the number of configurations reachable from an input.
Reachability

- **Lemma**: For any nondeterministic linear bounded automaton there is another which can compute the number of configurations reachable from an input.

- **Proof**: we begin with an arbitrary machine which has k instructions, one track, s symbols, and an input of length n. We shall iteratively count the number of configurations \( n_i \) reachable from the initial configuration.
Reachability algorithm

\[ n_0 = 1, \quad i = 0 \]
repeat
\[ i = i + 1 \]
\[ n_i = 0, \quad m := 0 \]
for \( j = 1 \) to \( k \times n \times s^n \)
generate \( C_j \)
guess whether \( C_j \) can be reached in \( i \) steps or less
if path from \( C_0 \) to \( C_j \) is verifiable then
\[ n_i = n_i + 1 \]
if reached in less than \( i \) steps then \( m = m + 1 \)
verify that \( m = n_i - 1 \) (otherwise reject)
until \( n_i = n_{i-1} \)
Reachability algorithm

The guessing step is just the algorithm of our last lemma. We do it by finding all of the configurations reachable in less than \( i \) steps and seeing if any of them is \( C_j \) or if one more step will produce \( C_j \). Since we know \( n \geq i \), we can verify that we have looked at all of them.
Reachability algorithm

- The guessing step is just the algorithm of our last lemma
Reachability algorithm

- The guessing step is just the algorithm of our last lemma.
- We do it by finding all of the configurations reachable in less than \( i \) steps and seeing if any of them is \( C_j \) or if one more step will produce \( C_j \).
The guessing step is just the algorithm of our last lemma.

We do it by finding all of the configurations reachable in less than $i$ steps and seeing if any of them is $C_j$ or if one more step will produce $C_j$.

Since we know $n_{i-1}$, we can verify that we have looked at all of them.
Reachability

The class of sets accepted by nondeterministic linear bounded automata is closed under complement.

Proof
To build a machine which accepts the complement of the set accepted by some NLBA involves putting the previous two together. First find out exactly how many configurations are reachable. Then examine all of them and if any halting configurations are encountered, reject. Otherwise accept.

By Aravind and Omesh Guide: Deepak D'Souza
The class of sets accepted by nondeterministic linear bounded automata is closed under complement.
Reachability

- The class of sets accepted by nondeterministic linear bounded automata is closed under complement.
- Proof
**Reachability**

- The class of sets accepted by nondeterministic linear bounded automata is closed under complement.

**Proof**

To build a machine which accepts the complement of the set accepted by some nLBA involves putting the previous two together. First find out exactly how many configurations are reachable. Then examine all of them and if any halting configurations are encountered, reject. Otherwise accept.
Union and Intersection
Union and Intersection

- Union
Union and Intersection

- Union
- Intersection
Thank You

THANK YOU !!!

Any Questions ?
(added) $\text{CSG} = \text{LBA}$

- A language is accepted by an LBA iff it is generated by a CSG
- Just like equivalence between CFG and PDA
- Given an $x \in \text{CSG} G$, you can intuitively see that and LBA can start with $S$, and nondeterministically choose all derivations from $S$ and see if they are equal to the input string $x$. Because CSLs are non-contracting, the LBA only needs to generate derivations of length $\leq |x|$. This is because if it generates a derivation longer than $|x|$, it will never be able to shrink to the size of $|x|$.
(added) CSG TO LBA
(added) CSG TO LBA

- M traverses the tape from left to right, using its finite state memory to search for a subword equal to the left hand side of some production of G. For this, a number of last read tape symbols remain stored in the state memory of M.
(added) CSG TO LBA

- M traverses the tape from left to right, using its finite state memory to search for a subword equal to the left hand side of some production of G. For this, a number of last read tape symbols remain stored in the state memory of M.

- After finding in this way a subword equal to left hand side of a production, M decides whether or not it will use this production in the simulation (nondeterminism). If not, M continues its search.
(added) CSG TO LBA
If M decides to use in the simulation the production it found, it next conducts the simulation of the corresponding direct derivation by G. If this leads to lengthening of the word, i.e., the right hand side of the production is longer than the left hand side, a suffix of the word already derived must be moved to the right respectively, if possible. This takes a lot of transitions. If there is not sufficient space available, that is, the derived word is too long and there would be an overflow, M halts in a nonterminal state. After simulating the direct derivation step, M again starts its search from the left endmarker.
(added) CSG TO LBA
(added) CSG TO LBA

If M decides to use in the simulation the production it found, it next conducts the If there is no way of continuing the simulated derivation, that is, no applicable production is found, M checks whether or not the derived word equals the input, stored in the lower track for this purpose, and in the positive case halts in a terminal state. In the negative case, M halts in a nonterminal state.