Lecture 4 – Outline

So far we saw
Lecture 4 – Outline

So far we saw

- finite automata,
- regular languages,
- regular expressions,
- pumping lemma for regular languages.

Today, we introduce stronger machines and languages with more expressive power:

- pushdown automata,
- context-free languages,
- context-free grammars,
- pumping lemma for context-free languages.
Lecture 4 – Outline

So far we saw

- finite automata,
- regular languages,
Lecture 4 – Outline

So far we saw

- finite automata,
- regular languages,
- regular expressions,
Lecture 4 – Outline

So far we saw
- finite automata,
- regular languages,
- regular expressions,
- pumping lemma for regular languages.
Lecture 4 – Outline

So far we saw
- finite automata,
- regular languages,
- regular expressions,
- pumping lemma for regular languages.

Today, we introduce stronger machines and languages with more expressive power:
Lecture 4 – Outline

So far we saw

- finite automata,
- regular languages,
- regular expressions,
- pumping lemma for regular languages.

Today, we introduce stronger machines and languages with more expressive power:

- pushdown automata,
Lecture 4 – Outline

So far we saw

- finite automata,
- regular languages,
- regular expressions,
- pumping lemma for regular languages.

Today, we introduce stronger machines and languages with more expressive power:

- pushdown automata,
- context-free languages,
Lecture 4 – Outline

So far we saw
- finite automata,
- regular languages,
- regular expressions,
- pumping lemma for regular languages.

Today, we introduce stronger machines and languages with more expressive power:
- pushdown automata,
- context-free languages,
- context-free grammars,
Lecture 4 – Outline

So far we saw

- finite automata,
- regular languages,
- regular expressions,
- pumping lemma for regular languages.

Today, we introduce stronger machines and languages with more expressive power:

- pushdown automata,
- context-free languages,
- context-free grammars,
- pumping lemma for context-free languages.
Context-Free Grammars

This is an example of a context free grammar, $G_1$:

- $A \rightarrow 0A1$
- $A \rightarrow B$
- $B \rightarrow \#$

Terminology: Each line is a substitution rule or production. Each rule has the form: symbol $\rightarrow$ string. The left-hand symbol is a variable (usually upper-case). A string consists of variables and terminals. One variable is the start variable.
Context-Free Grammars

This is an example of a context free grammar, $G_1$:

- $A \rightarrow 0A1$
- $A \rightarrow B$
- $B \rightarrow #$

Terminology:

- Each line is a substitution rule or production.
- Each rule has the form: symbol $\rightarrow$ string. The left-hand symbol is a variable (usually upper-case).
Context-Free Grammars

This is an example of a context free grammar, $G_1$:

- $A \rightarrow 0A1$
- $A \rightarrow B$
- $B \rightarrow #$

Terminology:

- Each line is a substitution rule or production.
- Each rule has the form: symbol $\rightarrow$ string. The left-hand symbol is a variable (usually upper-case).
- A string consists of variables and terminals.

Slides modified by Benny Chor, based on original slides by Maurice Herlihy, Brown University.
Context-Free Grammars

This is an example of a context free grammar, $G_1$:

- $A \rightarrow 0A1$
- $A \rightarrow B$
- $B \rightarrow \#$

Terminology:

- Each line is a substitution rule or production.
- Each rule has the form: symbol $\rightarrow$ string. The left-hand symbol is a variable (usually upper-case).
- A string consists of variables and terminals.
- One variable is the start variable.
Rules for Generating Strings

- Write down the start variable (lhs of top rule).
Rules for Generating Strings

- Write down the start variable (lhs of top rule).
- Pick a variable written down in current string and a derivation that starts with that variable.
Rules for Generating Strings

1. Write down the start variable (lhs of top rule).
2. Pick a variable written down in current string and a derivation that starts with that variable.
3. Replace that variable with right-hand side of that derivation.
Rules for Generating Strings

- Write down the start variable (lhs of top rule).
- Pick a variable written down in current string and a derivation that starts with that variable.
- Replace that variable with right-hand side of that derivation.
- Repeat until no variables remain.
Rules for Generating Strings

- Write down the start variable (lhs of top rule).
- Pick a variable written down in current string and a derivation that starts with that variable.
- Replace that variable with right-hand side of that derivation.
- Repeat until no variables remain.
- Return final string (concatenation of terminals).
Example

Grammar $G_1$:

- $A \rightarrow 0A1$
- $A \rightarrow B$
- $B \rightarrow \#$

Derivation with $G_1$:

$A \Rightarrow 0A1$
$A1 \Rightarrow 00A1$
$A11 \Rightarrow 000A1$
$A111 \Rightarrow 000B111$

Slides modified by Benny Chor, based on original slides by Maurice Herlihy, Brown University.
Example

Grammar $G_1$:

- $A \rightarrow 0A1$
- $A \rightarrow B$
- $B \rightarrow \#$

Derivation with $G_1$:

\[
\begin{align*}
A & \Rightarrow 0A1 \\
& \Rightarrow 00A11 \\
& \Rightarrow 000A111 \\
& \Rightarrow 000B111 \\
& \Rightarrow 000\#111
\end{align*}
\]
A Parse Tree

What strings can be generated in this way from the grammar $G$?

Exactly those of the form $0^n # 1^m$ ($n \geq 0$).
A Parse Tree

Question: What strings can be generated in this way from the grammar $G_1$?
A Parse Tree

**Question:** What strings can be generated in this way from the grammar $G_1$?

**Answer:** Exactly those of the form $0^n \#1^n$ ($n \geq 0$).
Context-Free Languages

The language generated in this way is the language of the grammar.
Context-Free Languages

The language generated in this way is the language of the grammar.

For example, $L(G_1)$ is $\{0^n \#1^n | n \geq 0\}$. 
Context-Free Languages

The language generated in this way is the language of the grammar.

For example, \( L(G_1) \) is \( \{0^n \# 1^n | n \geq 0\} \).

Any language generated by a context-free grammar is called a context-free language.
A Useful Abbreviation

Rules with same variable on left hand side

\[
A \rightarrow 0A1 \\
A \rightarrow B
\]

are written as:
A Useful Abbreviation

Rules with same variable on left hand side

\[
\begin{align*}
A & \rightarrow 0A1 \\
A & \rightarrow B
\end{align*}
\]

are written as:

\[
A \rightarrow 0A1 \mid B
\]
English-like Sentences

A grammar $G_2$ to describe a few English sentences:

- $<\text{SENTENCE}> \rightarrow <\text{NOUN-PHRASE}> <\text{VERB}>
- $<\text{NOUN-PHRASE}> \rightarrow <\text{ARTICLE}> <\text{NOUN}>
- $<\text{NOUN}> \rightarrow \text{boy} | \text{girl} | \text{flower}
- $<\text{ARTICLE}> \rightarrow \text{a} | \text{the}
- $<\text{VERB}> \rightarrow \text{touches} | \text{likes} | \text{sees}$
Deriving English-like Sentences

A specific derivation in $G_2$:

\[
\langle \text{SENTENCE} \rangle \Rightarrow \langle \text{NOUN-PHRASE} \rangle \langle \text{VERB} \rangle \\
\Rightarrow \langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle \langle \text{VERB} \rangle \\
\Rightarrow \text{a} \langle \text{NOUN} \rangle \langle \text{VERB} \rangle \\
\Rightarrow \text{a boy} \langle \text{VERB} \rangle \\
\Rightarrow \text{a boy sees}
\]
Deriving English-like Sentences

A specific derivation in $G_2$:

\[
\begin{align*}
&\langle \text{SENTENCE} \rangle \Rightarrow \langle \text{NOUN-PHRASE} \rangle \langle \text{VERB} \rangle \\
&\quad \Rightarrow \langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle \langle \text{VERB} \rangle \\
&\quad \Rightarrow \langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle \langle \text{VERB} \rangle \\
&\quad \Rightarrow a \langle \text{NOUN} \rangle \langle \text{VERB} \rangle \\
&\quad \Rightarrow a \text{ boy} \langle \text{VERB} \rangle \\
&\quad \Rightarrow a \text{ boy sees}
\end{align*}
\]

More strings in $G_2$:

- a flower sees
- the girl touches
Derivation and Parse Tree

\[
\begin{align*}
\langle \text{SENTENCE} \rangle & \Rightarrow \langle \text{NOUN-PHRASE} \rangle \langle \text{VERB} \rangle \\
& \Rightarrow \langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle \langle \text{VERB} \rangle \\
& \Rightarrow \text{a} \ \langle \text{NOUN} \rangle \langle \text{VERB} \rangle \\
& \Rightarrow \text{a boy} \ \langle \text{VERB} \rangle \\
& \Rightarrow \text{a boy sees}
\end{align*}
\]
Formal Definitions

A context-free grammar is a 4-tuple \((V, \Sigma, R, S)\) where

- \(V\) is a finite set of variables,
Formal Definitions

A context-free grammar is a 4-tuple \((V, \Sigma, R, S)\) where

- \(V\) is a finite set of variables,
- \(\Sigma\) is a finite set of terminals,
Formal Definitions

A context-free grammar is a 4-tuple \((V, \Sigma, R, S)\) where

- \(V\) is a finite set of variables,
- \(\Sigma\) is a finite set of terminals,
- \(R\) is a finite set of rules: each rule is a variable and a finite string of variables and terminals.
Formal Definitions

A context-free grammar is a 4-tuple \((V, \Sigma, R, S)\) where

- \(V\) is a finite set of variables,
- \(\Sigma\) is a finite set of terminals,
- \(R\) is a finite set of rules: each rule is a variable and a finite string of variables and terminals.
- \(S\) is the start symbol.
Formal Definitions

If \( u \) and \( v \) are strings of variables and terminals,
Formal Definitions

- If $u$ and $v$ are strings of variables and terminals,
- and $A \rightarrow w$ is a rule of the grammar, then
Formal Definitions

- If $u$ and $v$ are strings of variables and terminals,
- and $A \rightarrow w$ is a rule of the grammar, then
- we say $uAv$ yields $uwv$, written $uAv \Rightarrow uwv$. 
Formal Definitions

- If $u$ and $v$ are strings of variables and terminals,
- and $A \rightarrow w$ is a rule of the grammar, then
- we say $uAv$ yields $uwv$, written $uAv \Rightarrow uwv$.

We write $u \Rightarrow^* v$ if $u = v$ or

$$u \Rightarrow u_1 \Rightarrow \ldots \Rightarrow u_k \Rightarrow v.$$  

for some sequence $u_1, u_2, \ldots, u_k$. 
Formal Definitions

- If $u$ and $v$ are strings of variables and terminals, and $A \rightarrow w$ is a rule of the grammar, then we say $uAv$ yields $uwv$, written $uAv \Rightarrow uwv$.

We write $u \Rightarrow^* v$ if $u = v$ or

$$u \Rightarrow u_1 \Rightarrow \ldots \Rightarrow u_k \Rightarrow v.$$  

for some sequence $u_1, u_2, \ldots, u_k$.

Definition: The language of the grammar is

$$\left\{ w \mid S \Rightarrow^* w \right\}.$$
Example

Consider $G_4 = (V, \{a, b\}, R, S)$.

$R$ (Rules): $S \rightarrow aSb \mid SS \mid \varepsilon$. 
Example

Consider $G_4 = (V, \{a, b\}, R, S)$.

$R$ (Rules): $S \rightarrow aSb | SS | \varepsilon$.

Some words in the language: $aabb, aababb$. 

But what is this language?

Hint: Think of parentheses.
Example

Consider $G_4 = (V, \{a, b\}, R, S)$.

$R$ (Rules): $S \rightarrow aSb | SS | \varepsilon$.

Some words in the language: $aabb, aababb$.

Q.: But what is this language?
Example

Consider $G_4 = (V, \{a, b\}, R, S)$.

$R$ (Rules): $S \rightarrow aSb \mid SS \mid \varepsilon$.

Some words in the language: $aabb, aababb$.

Q.: But what is this language?

Hint: Think of parentheses.
Arythmetic Example

Consider \((V, \Sigma, R, E)\) where

- \(V = \{ E, T, F \}\)
- \(\Sigma = \{ a, +, \times, (, ) \}\)

Rules:

- \(E \rightarrow E + T \mid T\)
- \(T \rightarrow T \times F \mid F\)
- \(F \rightarrow (E) \mid a\)

Strings generated by the grammar: 

\(a + a \times a\) and \((a + a) \times a\).

What is the language of this grammar?

Hint: arithmetic expressions.

\(E = \) expression, \(T = \) term, \(F = \) factor.
Arythmetic Example

Consider \((V, \Sigma, R, E)\) where

- \(V = \{E, T, F\}\)
- \(\Sigma = \{a, +, \times, (, )\}\)

Rules:

- \(E \rightarrow E + T \mid T\)
- \(T \rightarrow T \times F \mid F\)
- \(F \rightarrow (E) \mid a\)

Strings generated by the grammar:

\(a + a \times a\) and \((a + a) \times a\).
Arythmetic Example

Consider \((V, \Sigma, R, E)\) where

- \(V = \{ E, T, F \}\)
- \(\Sigma = \{ a, +, \times, (, ) \}\)

Rules:

- \(E \rightarrow E + T \mid T\)
- \(T \rightarrow T \times F \mid F\)
- \(F \rightarrow (E) \mid a\)

Strings generated by the grammar:

- \(a + a \times a\) and \((a + a) \times a\).

What is the language of this grammar?

Hint: arithmetic expressions.

\(E = \text{expression}, T = \text{term}, F = \text{factor}.\)
Parse Tree for $a + a \times a$

\[
E \rightarrow E + T \mid T \\
T \rightarrow T \times F \mid F \\
F \rightarrow (E) \mid a
\]
Parse Tree for \( a + a \times a \)

\[
E \rightarrow E + T | T \\
T \rightarrow T \times F | F \\
F \rightarrow (E) | a
\]
Parse Tree for \((a + a) \times a\)

\[
\begin{align*}
E & \rightarrow E + T \mid T \\
T & \rightarrow T \times F \mid F \\
F & \rightarrow (E) \mid a
\end{align*}
\]
Parse Tree for \((a + a) \times a\)

\[
E \rightarrow E + T \mid T \\
T \rightarrow T \times F \mid F \\
F \rightarrow (E) \mid a
\]
Designing Context-Free Grammars

No recipe in general, but few rules-of-thumb

- If CFG is the **union** of several CFGs, rename variables (**not terminals**) so they are disjoint, and add new rule $S \rightarrow S_1 \mid S_2 \mid \ldots \mid S_i$. 

For languages with **linked** substrings (like \{0^n #1^n | n ≥ 0\}), a rule of form $R \rightarrow uRv$ may be helpful, forcing desired relation between substrings.
Designing Context-Free Grammars

No recipe in general, but few rules-of-thumb

- If CFG is the union of several CFGs, rename variables (not terminals) so they are disjoint, and add new rule $S \rightarrow S_1 \mid S_2 \mid \ldots \mid S_i$.

- To construct CFG for a regular language, “follow” a DFA for the language. For initial state $q_0$, make $R_0$ the start variable. For state transition $\delta(q_i, a) = q_j$ add rule $R_i \rightarrow aR_j$ to grammar. For each final state $q_f$, add rule $R_f \rightarrow \varepsilon$ to grammar.
Designing Context-Free Grammars

No recipe in general, but few rules-of-thumb

- If CFG is the union of several CFGs, rename variables (not terminals) so they are disjoint, and add new rule $S \rightarrow S_1 \mid S_2 \mid \ldots \mid S_i$.

- To construct CFG for a regular language, “follow” a DFA for the language. For initial state $q_0$, make $R_0$ the start variable. For state transition $\delta(q_i, a) = q_j$ add rule $R_i \rightarrow aR_j$ to grammar. For each final state $q_f$, add rule $R_f \rightarrow \varepsilon$ to grammar.

- For languages with linked substrings (like $\{0^n \#1^n \mid n \geq 0\}$), a rule of form $R \rightarrow uRv$ may be helpful, forcing desired relation between substrings.
Closure Properties

- Regular languages are closed under
  - union
  - concatenation
  - star
Closure Properties

- Regular languages are closed under
  - union
Closure Properties

- Regular languages are closed under
  - union
  - concatenation
Closure Properties

- Regular languages are closed under
  - union
  - concatenation
  - star
Closure Properties

- Regular languages are closed under
  - union
  - concatenation
  - star
- Context-Free Languages are closed under
Closure Properties

- Regular languages are closed under
  - union
  - concatenation
  - star

- Context-Free Languages are closed under
  - union: $S \rightarrow S_1 \mid S_2$
Closure Properties

- Regular languages are closed under
  - union
  - concatenation
  - star

- Context-Free Languages are closed under
  - union: \( S \rightarrow S_1 \mid S_2 \)
  - concatenation \( S \rightarrow S_1S_2 \)
Closure Properties

- Regular languages are closed under
  - union
  - concatenation
  - star

- Context-Free Languages are closed under
  - union: $S \rightarrow S_1 \mid S_2$
  - concatenation $S \rightarrow S_1 S_2$
  - star $S \rightarrow \varepsilon \mid SS$
More Closure Properties

- Regular languages are also closed under

What about complement and intersection of context-free languages? Not clear...
More Closure Properties

- Regular languages are also closed under complement (reverse accept/non-accept states of DFA)
More Closure Properties

- Regular languages are also closed under complement (reverse accept/non-accept states of DFA)
- Intersection \( L_1 \cap L_2 = \overline{L_1} \cup \overline{L_2} \).
More Closure Properties

- Regular languages are also closed under complement (reverse accept/non-accept states of DFA)

- intersection \(L_1 \cap L_2 = \overline{L_1} \cup \overline{L_2}\).

- What about complement and intersection of context-free languages?
More Closure Properties

- Regular languages are also closed under complement (reverse accept/non-accept states of DFA)

  intersection \( (L_1 \cap L_2 = \overline{L_1} \cup \overline{L_2}) \).

- What about complement and intersection of context-free languages?
  - Not clear . . .
Ambiguity

Grammar: $E \rightarrow E + E \mid E \times E \mid (E) \mid a$

Slides modified by Benny Chor, based on original slides by Maurice Herlihy, Brown University.
Ambiguity

We say that a string $w$ is derived *ambiguously* from grammer $G$ if $w$ has two or more parse trees that generate it from $G$. 
Ambiguity

We say that a string $w$ is derived ambiguously from grammar $G$ if $w$ has two or more parse trees that generate it from $G$.

Ambiguity is usually not only a syntactic notion but also a semantic one, implying multiple meanings for the same string.
Ambiguity

We say that a string $w$ is derived ambiguously from grammer $G$ if $w$ has two or more parse trees that generate it from $G$.

Ambiguity is usually not only a syntactic notion but also a semantic one, implying multiple meanings for the same string.

It is sometime possible to eliminate ambiguity by finding a different context free grammer generating the same language. This is true for the grammer above, which can be replaced by unambiguous grammer from slide (14).
Ambiguity

We say that a string $w$ is derived ambiguously from grammar $G$ if $w$ has two or more parse trees that generate it from $G$.

Ambiguity is usually not only a syntactic notion but also a semantic one, implying multiple meanings for the same string.

It is sometime possible to eliminate ambiguity by finding a different context free grammar generating the same language. This is true for the grammar above, which can be replaced by unambiguous grammar from slide (14).

Some languages (e.g. $\{1^i2^j3^k \mid i = j \text{ or } j = k\}$) are inherently ambiguous.
Chomsky Normal Form

A simplified, canonical form of context free grammers.
Every rule has the form

\[
\begin{align*}
A & \rightarrow BC \\
A & \rightarrow a \\
S & \rightarrow \varepsilon
\end{align*}
\]

where \( S \) is the start symbol, \( A, B \) and \( C \) are any variable, except \( B \) and \( C \) not the start symbol.
Theorem

**Theorem:** Any context-free language is generated by a context-free grammar in Chomsky normal form.

Basic idea:

- Add new start symbol $S_0$. 
Theorem

**Theorem:** Any context-free language is generated by a context-free grammar in Chomsky normal form.

Basic idea:

- Add new start symbol $S_0$.
- Eliminate all $\varepsilon$ rules of the form $A \rightarrow \varepsilon$. 
Theorem

Theorem: Any context-free language is generated by a context-free grammar in Chomsky normal form.

Basic idea:

- Add new start symbol $S_0$.
- Eliminate all $\varepsilon$ rules of the form $A \rightarrow \varepsilon$.
- Eliminate all “unit” rules of the form $A \rightarrow B$. 

Theorem

**Theorem:** Any context-free language is generated by a context-free grammar in Chomsky normal form.

Basic idea:

- Add new start symbol $S_0$.
- Eliminate all $\varepsilon$ rules of the form $A \rightarrow \varepsilon$.
- Eliminate all “unit” rules of the form $A \rightarrow B$.
- Patch up rules so that grammar generates the same language.
Theorem

**Theorem:** Any context-free language is generated by a context-free grammar in Chomsky normal form.

Basic idea:

- Add new start symbol $S_0$.
- Eliminate all $\varepsilon$ rules of the form $A \rightarrow \varepsilon$.
- Eliminate all “unit” rules of the form $A \rightarrow B$.
- Patch up rules so that grammar generates the same language.
- Convert remaining long rules to proper form.
Proof

Add new start symbol $S_0$ and rule $S_0 \rightarrow S$.
Guarantees that new start symbol does not appear on right-hand-side of a rule.
Proof

Eliminating $\varepsilon$ rules.

Repeat:

- remove some $A \rightarrow \varepsilon$. 

...
Proof

Eliminating $\varepsilon$ rules.

Repeat:

- remove some $A \rightarrow \varepsilon$.
- for each $R \rightarrow uAv$, add rule $R \rightarrow uv$. 

and so on: for $R \rightarrow uAvAw$ add $R \rightarrow uvAw$, $R \rightarrow uAvw$, and $R \rightarrow uvw$. 

for $R \rightarrow A$ add $R \rightarrow \varepsilon$, except if $R \rightarrow \varepsilon$ has already been removed.

until all $\varepsilon$-rules not involving the start variable have been removed.
Proof

Eliminating ε rules.

Repeat:

- remove some \( A \rightarrow \varepsilon \).
- for each \( R \rightarrow uAv \), add rule \( R \rightarrow uv \).
- and so on: for \( R \rightarrow uAvAw \) add \( R \rightarrow uvAw \), \( R \rightarrow uAvw \), and \( R \rightarrow uvw \).
Proof

Eliminating $\varepsilon$ rules.

Repeat:

- remove some $A \rightarrow \varepsilon$.
- for each $R \rightarrow uAv$, add rule $R \rightarrow uv$.
- and so on: for $R \rightarrow uAvAw$ add $R \rightarrow uvAw$, $R \rightarrow uAvw$, and $R \rightarrow uvw$.
- for $R \rightarrow A$ add $R \rightarrow \varepsilon$, except if $R \rightarrow \varepsilon$ has already been removed.
Proof

Eliminating $\varepsilon$ rules.

Repeat:

- remove some $A \rightarrow \varepsilon$.
- for each $R \rightarrow uAv$, add rule $R \rightarrow uv$.
- and so on: for $R \rightarrow uAvAw$ add $R \rightarrow uvAw$, $R \rightarrow uAvw$, and $R \rightarrow uvw$.
- for $R \rightarrow A$ add $R \rightarrow \varepsilon$, except if $R \rightarrow \varepsilon$ has already been removed.

until all $\varepsilon$-rules not involving the start variable have been removed.
Proof

Eliminate unit rules.

Repeat:

- remove some $A \rightarrow B$. 
Proof

Eliminate unit rules.

Repeat:

- remove some $A \rightarrow B$.
- for each $B \rightarrow u$, add rule $A \rightarrow u$, unless this is previously removed unit rule. ($u$ is a string of variables and terminals.)
Proof

Eliminate unit rules.

Repeat:

- remove some $A \rightarrow B$.
- for each $B \rightarrow u$, add rule $A \rightarrow u$, unless this is previously removed unit rule. ($u$ is a string of variables and terminals.)

until all unit rules have been removed.
Proof

Finally, convert long rules.
To replace each $A \rightarrow u_1 u_2 \ldots u_k$ (for $k \geq 3$), introduce new non-terminals

$$N_1, N_2, \ldots, N_{k-1}$$

and rules
Proof

Finally, convert long rules.
To replace each $A \rightarrow u_1u_2 \ldots u_k$ (for $k \geq 3$), introduce new non-terminals

$$N_1, N_2, \ldots, N_{k-1}$$

and rules

$$A \rightarrow u_1 N_1$$
$$N_1 \rightarrow u_2 N_2$$
$$\vdots$$
$$N_{k-3} \rightarrow u_{k-2} N_{k-2}$$
$$N_{k-2} \rightarrow u_{k-1} u_k$$

Slides modified by Benny Chor, based on original slides by Maurice Herlihy, Brown University.
Conversion Example

Initial Grammar:

\[
S \rightarrow ASA \mid aB \\
A \rightarrow B \mid S \\
B \rightarrow b \mid \varepsilon
\]

(1) Add new start state:

\[
S_0 \rightarrow S \\
S \rightarrow ASA \mid aB \\
A \rightarrow B \mid S \\
B \rightarrow b \mid \varepsilon
\]
Conversion Example (2)

\[
\begin{align*}
S_0 & \to S \\
S & \to ASA \mid aB \\
A & \to B \mid S \\
B & \to b \mid \varepsilon
\end{align*}
\]

(2) Remove \( \varepsilon \)-rule \( B \to \varepsilon \):

\[
\begin{align*}
S_0 & \to S \\
S & \to ASA \mid aB \mid a \\
A & \to B \mid S \mid \varepsilon \\
B & \to b \mid \varepsilon
\end{align*}
\]
Conversion Example (3)

\[
\begin{align*}
  S_0 & \rightarrow S \\
  S & \rightarrow ASA | aB | a \\
  A & \rightarrow B | S | \varepsilon \\
  B & \rightarrow b
\end{align*}
\]

(3) Remove \( \varepsilon \)-rule \( A \rightarrow \varepsilon \):

\[
\begin{align*}
  S_0 & \rightarrow S \\
  S' & \rightarrow ASA | aB | a | AS | SA | S \\
  A & \rightarrow B | S | \varepsilon \\
  B & \rightarrow b
\end{align*}
\]
Conversion Example (4)

\[
\begin{align*}
S_0 & \rightarrow S \\
S & \rightarrow ASA | aB | a | AS | SA | S \\
A & \rightarrow B | S \\
B & \rightarrow b 
\end{align*}
\]

(4) Remove unit rule \( S \rightarrow S \)

\[
\begin{align*}
S_0 & \rightarrow S \\
S' & \rightarrow ASA | aB | a | AS | SA | S \\
A & \rightarrow B | S \\
B & \rightarrow b 
\end{align*}
\]
Conversion Example (5)

\[
\begin{align*}
S_0 & \rightarrow S \\
S & \rightarrow ASA | aB | a | AS | SA \\
A & \rightarrow B | S \\
B & \rightarrow b
\end{align*}
\]

(5) Remove unit rule \( S_0 \rightarrow S \):

\[
\begin{align*}
S_0 & \rightarrow S | ASA | aB | a | AS | SA \\
S & \rightarrow ASA | aB | a | AS | SA \\
A & \rightarrow B | S \\
B & \rightarrow b
\end{align*}
\]
Conversion Example (6)

\[ S_0 \rightarrow ASA | aB | a | AS | SA \]
\[ S \rightarrow ASA | aB | a | AS | SA \]
\[ A \rightarrow B | S \]
\[ B \rightarrow b \]

(6) Remove unit rule \( A \rightarrow B \):

\[ S_0 \rightarrow ASA | aB | a | AS | SA \]
\[ S \rightarrow ASA | aB | a | AS | SA \]
\[ A \rightarrow B | S | b \]
\[ B \rightarrow b \]
Conversion Example (7)

\[ S_0 \rightarrow ASA \mid aB \mid a \mid AS \mid SA \]
\[ S \rightarrow ASA \mid aB \mid a \mid AS \mid SA \]
\[ A \rightarrow S \mid b \]
\[ B \rightarrow b \]

Remove unit rule \( A \rightarrow S \):

\[ S_0 \rightarrow ASA \mid aB \mid a \mid AS \mid SA \]
\[ S \rightarrow ASA \mid aB \mid a \mid AS \mid SA \]
\[ A \rightarrow S \mid b \mid ASA \mid aB \mid a \mid AS \mid SA \]
\[ B \rightarrow b \]
Conversion Example (8)

\[
\begin{align*}
S_0 & \rightarrow ASA | aB | a | AS | SA \\
S & \rightarrow ASA | aB | a | AS | SA \\
A & \rightarrow b | ASA | aB | a | AS | SA \\
B & \rightarrow b \\
\end{align*}
\]

(8) Final simplification – treat long rules:

\[
\begin{align*}
S_0 & \rightarrow AA_1 | UB | a | SA | AS \\
S & \rightarrow AA_1 | UB | a | SA | AS \\
A & \rightarrow b | AA_1 | UB | a | SA | AS \\
A_1 & \rightarrow SA \\
U & \rightarrow a \\
B & \rightarrow b \\
\end{align*}
\]