COMP2022: Formal Languages and Logic

2017, Semester 1, Week 6

Joseph Godbehere

Adapted from slides by A/Prof Kalina Yacef

April 11, 2017

THE UNIVERSITY OF SYDNEY

Announcements

Friday 14th April is a public holiday

- No scheduled office hour / advanced seminar

Tuesday 25th April is a public holiday

- Replacement lecture:
  - 4pm – 6pm on Wednesday 26th April
  - PNR LT1 (i.e. the LT *next* to this one)
- Tuesday tutorials:
  - Join Wednesday / Thursday tutorials *if there is space*
    - See Ed for times/locations
  - Otherwise work through the exercises independently
- No quiz week 7
- Quiz week 8 will assess content from weeks 6 and/or 7

Outline

- LL(k) Table-Descent Parsing (continued)

- Grammar transformations
  - Finding equivalent LL(1) grammars
  - Chomsky Normal Form
  - Greibach Normal Form

FIRST and FOLLOW sets

In order to fill in the entries of the table-drived parser, we need to compute some FIRST and FOLLOW sets.

$FIRST(\alpha)$ is the set of all terminals which could start strings derived from α. We will need to calculate these for every production of G (i.e. the *right hand side* of each rule).

$FOLLOW(V)$ is the set of all terminals which which could follow the variable V at any stage of the derivation. Needed whenever V can derive ε.

Why do we need them?

Let the current input symbol be b, and the top of the stack be X.

There are two ways we might try to derive a string starting with b:

1. Derive a string starting with b from the variable X. Look for a rule $X \to \alpha$ where $b \in FIRST(\alpha)$
2. If X can be derived to ε, then we might be able to derive a string starting with b by using the symbol(s) *following* X on the stack.
   i.e. If any of the production rules $X \to \alpha$ had $\varepsilon \in FIRST(\alpha)$, then we also look at $FOLLOW(X)$

Another way to calculate FIRST sets

Let $A \to X_1 ... X_n$ be a production of A, where X_i could be a terminal or variable.

We recursively compute $FIRST(X_1 ... X_n)$ by looking at X_1:

- If X_1 is a terminal symbol, then $FIRST(X_1 ... X_n) = \{X_1\}$
- If X_1 is a variable, then $FIRST(X_1 ... X_n)$ contains $FIRST(X_1) \setminus \{\varepsilon\}$
- If X_1 is a variable $\varepsilon \in FIRST(X_1)$ then $FIRST(X_1 ... X_n)$ also contains $FIRST(X_2 ... X_n)$

Don't forget that $FIRST(\varepsilon) = \{\varepsilon\}$, so if every X_i can generate ε, then rule 3 will (eventually) give us $\varepsilon \in FIRST(X_1 ... X_n)$

Another way to calculate FOLLOW sets

If $\varepsilon \in FIRST(\alpha)$ for some production rule $A \rightarrow \alpha$ then we need to compute $FOLLOW(A)$

Consider each production rule where A appears in the *right hand side*.

Let $V \rightarrow Y_1 ... Y_n A Z_1 ... Z_m$ (Y_i, Z_i can be terminals or variables)

- If A is the start symbol, then $\$ \in FOLLOW(A)$
- $FIRST(Z_1 ... Z_m) \setminus \{\varepsilon\} \subseteq FOLLOW(A)$
- If $\varepsilon \in FIRST(Z_1 ... Z_m)$ then $FOLLOW(V) \subseteq FOLLOW(A)$

EXAMPLES

See week 5

Constructing the parse table

Rows: one for each variable of the grammar

Columns: one for each terminal of the grammar, and for the end of string marker $

Steps to fill the table T:

1. If there is a rule $R \to \alpha$ with $b \in FIRST(\alpha)$ then put α in $T[R, b]$
2. If there is a rule $R \to \alpha$ with $\varepsilon \in FIRST(\alpha)$ and $b \in FOLLOW(R)$, then put α in $T[R, b]$

EXAMPLE

See week 5

Identify if a grammar is LL(1)

Recall that a grammar is LL(1) if it is sufficient to look at the next symbol to determine which rule to follow next.

i.e. If every cell in the LL(1) parse table contains at most one rule, then the grammar is LL(1)

More formally, a grammar is LL(1) iff for every variable A:

- Let $A \to \alpha_1 \mid ... \mid \alpha_n$ be the production rules for A
- Let $X_i = FIRST(\alpha_i)$ if $\varepsilon \notin FIRST(\alpha_i)$
- Let $X_i = FIRST(\alpha_i) \cup FOLLOW(A)$ otherwise
- Then $X_i \cap X_j = \emptyset$ for all $i \neq j$

Transforming non-LL(1) grammars

When a grammar is not LL(1) we try to find an equivalent grammar which is, by applying the following techniques:

- Left factoring
- Elimination of left recursion

Transforming non-LL(1) grammars

When a grammar is not LL(1) we try to find an equivalent grammar which is, by applying the following techniques:

- Left factoring
- Elimination of left recursion

Recall: grammars are *equivalent* if they generate the same language

Transforming non-LL(1) grammars

When a grammar is not LL(1) we try to find an equivalent grammar which is, by applying the following techniques:

- Left factoring
- Elimination of left recursion

Recall: grammars are *equivalent* if they generate the same language

Such a grammar does not always exist. For example no LL(k) grammar exists for the language
$\{a^n b^n \mid n \geq 0\} \cup \{a^n b^{2n} \mid n \geq 0\}$

Left factoring: why?

Consider the grammar fragment $S \rightarrow abcC \mid abdD$

- The two rules both start with the same prefix ab
- i.e. their FIRST sets both include a
- The LL(1) parse table will have multiple entries at (S, a).

Left factoring: why?

Consider the grammar fragment $S \to abcC \mid abdD$

- The two rules both start with the same prefix ab
- i.e. their FIRST sets both include a
- The LL(1) parse table will have multiple entries at (S, a).

We can "factor out" the string ab to obtain an equivalent grammar, where B is a new variable:

$$S \to abB$$
$$B \to cC \mid dD$$

This grammar fragment is equivalent, but is LL(1)

Left factoring: definition

If a string w appears on the left of several rules for a variable A:

$$A \to wX_1 \mid ... \mid wX_1$$

Then we can factor out w and introduce a new variable A':

$$\begin{aligned} A &\to wA' \\ A' &\to X_1 \mid ... \mid X_n \end{aligned}$$

Any other rules produced by A are unaffected.

Recursion

If a variable X can generate a string containing X itself, then it is recursive

Recursion

If a variable X can generate a string containing X itself, then it is recursive

- left-recursive: it occurs at the start of the string $X \Rightarrow^+ X\beta$

Recursion

If a variable X can generate a string containing X itself, then it is recursive

- left-recursive: it occurs at the start of the string $X \Rightarrow^{+} X\beta$
- right-recursive: it occurs at the end of the string $X \Rightarrow^{+} \alpha X$

Recursion

If a variable X can generate a string containing X itself, then it is recursive

- left-recursive: it occurs at the start of the string $X \Rightarrow^+ X\beta$
- right-recursive: it occurs at the end of the string $X \Rightarrow^+ \alpha X$
- self-embedding: it occurs in between: $X \Rightarrow^+ \alpha X \beta$

Recursion

If a variable X can generate a string containing X itself, then it is recursive

- left-recursive: it occurs at the start of the string $X \Rightarrow^+ X\beta$
- right-recursive: it occurs at the end of the string $X \Rightarrow^+ \alpha X$
- self-embedding: it occurs in between: $X \Rightarrow^+ \alpha X \beta$

A grammar is recursive if any of its variables is recursive

Recursion

If a variable X can generate a string containing X itself, then it is recursive

- left-recursive: it occurs at the start of the string $X \Rightarrow^+ X\beta$
- right-recursive: it occurs at the end of the string $X \Rightarrow^+ \alpha X$
- self-embedding: it occurs in between: $X \Rightarrow^+ \alpha X \beta$

A grammar is recursive if any of its variables is recursive

A grammar for an infinite language must contain at least one recursive variable

Eliminate left recursion: why?

Consider this simple grammar:

$$A \to c$$
$$A \to Ab$$

$$FIRST(c) = \{c\}$$
$$FIRST(Ab) = \{c\}$$

Eliminate left recursion: why?

Consider this simple grammar:

$$A \to c$$
$$A \to Ab$$

$$FIRST(c) = \{c\}$$
$$FIRST(Ab) = \{c\}$$

If we try to construct the parse table:

	b	c	\$
A		c or Ab	

ELIMINATE LEFT RECURSION: WHY?

Consider this simple grammar:

$$A \to c$$
$$A \to Ab$$

$$FIRST(c) = \{c\}$$
$$FIRST(Ab) = \{c\}$$

If we try to construct the parse table:

	b	c	\$
A		c or Ab	

The base cases for the recursion must have FIRST sets which intersect with the left recursive rule!

Eliminating left recursion

Let α, β be arbitrary strings of terminals and/or variables.
Let A be a variable, and R a new variable

If A has left recursive rules:

$$A \to A\alpha \mid \beta$$

It can be replaced with:

$$A \to \beta R$$
$$R \to \alpha R \mid \varepsilon$$

Eliminating left recursion

Let α, β be arbitrary strings of terminals and/or variables.
Let A be a variable, and R a new variable

If A has left recursive rules:

$$A \to A\alpha \mid \beta$$

It can be replaced with:

$$A \to \beta R$$
$$R \to \alpha R \mid \varepsilon$$

What do the parse trees look like for $\beta\alpha\alpha\alpha$ using the original and transformed grammar?

Simple example

$$A \to c$$
$$A \to Ab$$

Then $\alpha =$

Simple example

$$A \to c$$
$$A \to Ab$$

Then $\alpha = b$, $\beta =$

Simple example

$$A \to c$$
$$A \to Ab$$

Then $\alpha = b$, $\beta = c$, which gives us:

$$A \to cR$$
$$R \to bR \mid \varepsilon$$

Simple example

$$A \to c$$
$$A \to Ab$$

Then $\alpha = b$, $\beta = c$, which gives us:

$$A \to cR$$
$$R \to bR \mid \varepsilon$$

	b	c	\$
A		cR	
R	bR		ε

Complex example

$$E \to E + T$$
$$E \to E - T$$
$$E \to T$$
$$T \to a \mid b \mid c$$

Then $\alpha =$

Complex example

$$E \rightarrow E + T$$
$$E \rightarrow E - T$$
$$E \rightarrow T$$
$$T \rightarrow a \mid b \mid c$$

Then $\alpha = \ + T \mid -T$, $\beta =$

Complex example

$$
\begin{aligned}
E &\to E + T \\
E &\to E - T \\
E &\to T \\
T &\to a \mid b \mid c
\end{aligned}
$$

Then $\alpha = \ + T \mid -T$, $\beta = T$, which gives us:

$$
\begin{aligned}
E &\to TR \\
R &\to +TR \mid -TR \mid \varepsilon \\
T &\to a \mid b \mid c
\end{aligned}
$$

Complex example

$$E \to TR$$
$$R \to +TR \mid -TR \mid \varepsilon$$
$$T \to a \mid b \mid c$$

$FIRST(TR) =$

Complex example

$$E \to TR$$
$$R \to +TR \mid -TR \mid \varepsilon$$
$$T \to a \mid b \mid c$$

$FIRST(TR) = \{a, b, c\}$
$FIRST(+TR) =$

Complex example

$$E \to TR$$
$$R \to +TR \mid -TR \mid \varepsilon$$
$$T \to a \mid b \mid c$$

$FIRST(TR) = \{a, b, c\}$
$FIRST(+TR) = \{+\}$
$FIRST(-TR) =$

Complex example

$$E \to TR$$
$$R \to +TR \mid -TR \mid \varepsilon$$
$$T \to a \mid b \mid c$$

$FIRST(TR) = \{a, b, c\}$
$FIRST(+TR) = \{+\}$
$FIRST(-TR) = \{-\}$
Because $\varepsilon \in FIRST(\varepsilon)$, we calculate $FOLLOW(R) =$

Complex example

$$E \to TR$$
$$R \to +TR \mid -TR \mid \varepsilon$$
$$T \to a \mid b \mid c$$

$FIRST(TR) = \{a, b, c\}$
$FIRST(+TR) = \{+\}$
$FIRST(-TR) = \{-\}$
Because $\varepsilon \in FIRST(\varepsilon)$, we calculate $FOLLOW(R) = \{\$\}$

Complex example

$$E \to TR$$
$$R \to +TR \mid -TR \mid \varepsilon$$
$$T \to a \mid b \mid c$$

$FIRST(TR) = \{a, b, c\}$
$FIRST(+TR) = \{+\}$
$FIRST(-TR) = \{-\}$
Because $\varepsilon \in FIRST(\varepsilon)$, we calculate $FOLLOW(R) = \{\$\}$

	a	b	c	$+$	$-$	\$
E	TR	TR	TR			
R				$+TR$	$-TR$	ε
T	a	b	c			

Proving that a grammar is *not* LL(1)

It is sufficient to show any one of the following:

- The grammar is left recursive
- The grammar needs left factoring
- The first sets of the production rules for a variable are not disjoint

Typical exam question

Consider the grammar G:

$$S \to ST \mid ab$$
$$T \to aTbb \mid ab$$

Show that the grammar G is not LL(1)

Transform G to obtain a grammar G' which is LL(1)

Give the LL(1) parse table for G'

Chomsky Normal Form

Noam Chomsky, 1959

A grammar G is in Chomsky Normal Form iff every rule is of one of these forms:

- $A \to BC$ (neither B nor C are the start symbol)
- $A \to a$
- $S \to \varepsilon$ (iff S is the start symbol and $\varepsilon \in L(G)$)

Any CFG can be transformed into an equivalent Chomsky Normal Form.

We will use an algorithm which increases the size of the grammar to at most $|G|^2$

Chomsky Normal Form: but why??

Useful properties of CNF:

- No rules (except S) can produce ε.
- The derivations are always *binary trees*

The *CYK bottom up parsing algorithm* takes advantage of these

- It has a running time of $O(n^3 \cdot |G|)$, if G is in CNF
- i.e. effectively $O(n^3 \cdot |G|^2)$ if G was not in CNF (ignoring the one-off cost of performing the grammar transformations.)
- One of the most efficient parsers in terms of *worst case* asymptotic complexity.

Chomsky Normal Form: algorithm

1. **Eliminate the start symbol from all production rules**
2. Eliminate rules with non-solitary terminals
3. Eliminate rules with more than two variables
4. Eliminate epsilon productions
5. Eliminate unit rules

Add a new start symbol and rule $S_0 \to S$

Chomsky Normal Form: algorithm

1. Eliminate the start symbol from all production rules
2. **Eliminate rules with non-solitary terminals**
3. Eliminate rules with more than two variables
4. Eliminate epsilon productions
5. Eliminate unit rules

Create a new rule $N_a \to a$ for every terminal which appears in one or more rules which are not of the form $A \to a$, then replace a with N_a in every such production rule.

e.g. $A \to \alpha a \beta$ would become $A \to \alpha N_a \beta$

Chomsky Normal Form: algorithm

1. Eliminate the start symbol from all production rules
2. Eliminate rules with non-solitary terminals
3. **Eliminate rules with more than two variables**
4. Eliminate epsilon productions
5. Eliminate unit rules

For every rule of the form $A \rightarrow X_1 ... X_n$ (where $n > 2$), delete it and create new variables $A_1 ... A_{n-2}$ and rules:

$$
\begin{aligned}
A &\rightarrow X_1 A_1 \\
A_1 &\rightarrow X_2 A_2 \\
&\ldots \\
A_{n-3} &\rightarrow X_{n-2} A_{n-2} \\
A_{n-2} &\rightarrow X_{n-1} X_n
\end{aligned}
$$

Chomsky Normal Form: algorithm

1. Eliminate the start symbol from all production rules
2. Eliminate rules with non-solitary terminals
3. Eliminate rules with more than two variables
4. **Eliminate epsilon productions**
5. Eliminate unit rules

For every rule of the form $A \to \varepsilon$ (except $S_0 \to \varepsilon$)

- Delete the rule
- For each rule $R \to \alpha$ containing A, create every possible rule $R \to \alpha'$ where α' is α with one or more A's removed.

Example: removing $A \to \varepsilon$
If there is a rule $R \to \alpha A \beta A \gamma$ then we add 3 new rules
$R \to \alpha\beta\gamma, R \to \alpha A \beta\gamma, R \to \alpha\beta A\gamma$ (and keep the original)

Chomsky Normal Form: algorithm

1. Eliminate the start symbol from all production rules
2. Eliminate rules with non-solitary terminals
3. Eliminate rules with more than two variables
4. Eliminate epsilon productions
5. **Eliminate unit rules**

For each rule of the form $A \rightarrow B$

- For each rule of the form $B \rightarrow \alpha$ create a new rule $A \rightarrow \alpha$
- Remove the rule $A \rightarrow B$

Chomsky Normal Form: algorithm

1. Eliminate the start symbol from all production rules
2. Eliminate rules with non-solitary terminals
3. Eliminate rules with more than two variables
4. Eliminate epsilon productions
5. Eliminate unit rules

All done! The grammar should be equivalent to the original one, but in Chomsky Normal Form.

Chomsky Normal Form: example

$$S \to ASA \mid aB$$
$$A \to B \mid S$$
$$B \to b \mid \varepsilon$$

Eliminate start symbol from all production rules:

$$\mathbf{S_0} \to \mathbf{S}$$
$$S \to ASA \mid aB$$
$$A \to B \mid S$$
$$B \to b \mid \varepsilon$$

Chomsky Normal Form: example

$$
\begin{aligned}
S_0 &\to S \\
S &\to ASA \mid aB \\
A &\to B \mid S \\
B &\to b \mid \varepsilon
\end{aligned}
$$

Eliminate rules with non-solitary terminals:

$$
\begin{aligned}
S_0 &\to S \\
S &\to ASA \mid \mathbf{N_a}B \\
A &\to B \mid S \\
B &\to b \mid \varepsilon \\
\mathbf{N_a} &\to \mathbf{a}
\end{aligned}
$$

Chomsky Normal Form: example

$$
\begin{aligned}
S_0 &\rightarrow S \\
S &\rightarrow ASA \mid N_a B \\
A &\rightarrow B \mid S \\
B &\rightarrow b \mid \varepsilon \\
N_a &\rightarrow a
\end{aligned}
$$

Eliminate rules with more than two variables:

$$
\begin{aligned}
S_0 &\rightarrow S \\
S &\rightarrow \mathbf{AS_1} \mid N_a B \\
\mathbf{S_1} &\rightarrow \mathbf{SA} \\
A &\rightarrow B \mid S \\
B &\rightarrow b \mid \varepsilon \\
N_a &\rightarrow a
\end{aligned}
$$

Chomsky Normal Form: example

$$
\begin{aligned}
S_0 &\to S \\
S &\to AS_1 \mid N_a B \\
S_1 &\to SA \\
A &\to B \mid S \\
B &\to b \mid \varepsilon \\
N_a &\to a
\end{aligned}
$$

Eliminate epsilon production $B \to \varepsilon$

$$
\begin{aligned}
S_0 &\to S \\
S &\to AS_1 \mid N_a B \mid \mathbf{N_a} \\
S_1 &\to SA \\
A &\to B \mid S \mid \varepsilon \\
B &\to b \\
N_a &\to a
\end{aligned}
$$

Chomsky Normal Form: example

$$
\begin{aligned}
S_0 &\to S \\
S &\to AS_1 \mid N_a B \mid N_a \\
S_1 &\to SA \\
A &\to B \mid S \mid \varepsilon \\
B &\to b \\
N_a &\to a
\end{aligned}
$$

Eliminate epsilon production $A \to \varepsilon$

$$
\begin{aligned}
S_0 &\to S \\
S &\to AS_1 \mid N_a B \mid N_a \mid \mathbf{S_1} \\
S_1 &\to SA \mid \mathbf{S} \\
A &\to B \mid S \\
B &\to b \\
N_a &\to a
\end{aligned}
$$

Chomsky Normal Form: example

$$
\begin{aligned}
S_0 &\to S \\
S &\to AS_1 \mid N_a B \mid N_a \mid S_1 \\
S_1 &\to SA \mid S \\
A &\to B \mid S \\
B &\to b \\
N_a &\to a
\end{aligned}
$$

Eliminate unit rules $S \to N_a$, $A \to B$, then $S \to S_1$

$$
\begin{aligned}
S_0 &\to S \\
S &\to AS_1 \mid N_a B \mid \mathbf{a} \mid \mathbf{SA} \qquad \text{(and useless } S \to S\text{)} \\
S_1 &\to SA \mid S \\
A &\to \mathbf{b} \mid S \\
B &\to b \\
N_a &\to a
\end{aligned}
$$

Chomsky Normal Form: example

$$
\begin{aligned}
S_0 &\rightarrow S \\
S &\rightarrow AS_1 \mid N_aB \mid a \mid SA \\
S_1 &\rightarrow SA \mid S \\
A &\rightarrow b \mid S \\
B &\rightarrow b \\
N_a &\rightarrow a
\end{aligned}
$$

Eliminate unit rules $S_0 \rightarrow S$, $S_1 \rightarrow S$, $A \rightarrow S$

$$
\begin{aligned}
S_0 &\rightarrow \mathbf{AS_1} \mid \mathbf{N_aB} \mid \mathbf{a} \mid \mathbf{SA} \\
S &\rightarrow AS_1 \mid N_aB \mid a \mid SA \\
S_1 &\rightarrow \mathbf{AS_1} \mid \mathbf{N_aB} \mid \mathbf{a} \mid SA \\
A &\rightarrow b \mid \mathbf{AS_1} \mid \mathbf{N_aB} \mid \mathbf{a} \mid \mathbf{SA} \\
B &\rightarrow b \\
N_a &\rightarrow a
\end{aligned}
$$

Chomsky Normal Form: example

All done!

$$
\begin{aligned}
S_0 &\to AS_1 \mid N_aB \mid a \mid SA \\
S &\to AS_1 \mid N_aB \mid a \mid SA \\
S_1 &\to AS_1 \mid N_aB \mid a \mid SA \\
A &\to b \mid AS_1 \mid N_aB \mid a \mid SA \\
B &\to b \\
N_a &\to a
\end{aligned}
$$

Greibach Normal Form

Sheila Greibach, 1965

A grammar is in Greibach Normal Form iff every rule is in the form

- $A \to a$
- $A \to aB_1 ... B_n$ (where all B_i are variables)
- $S \to \varepsilon$ (iff $\varepsilon \in L(G)$)

Any CFG can be transformed into an equivalent Greibach Normal Form.

Greibach Normal Form: but why??

Useful properties of GNF:

- The grammar does not contain left recursion
- (but there can be many rules starting with the same terminal)
- *Every derivation step consumes one input symbol*

Greibach Normal Form: but why??

Useful properties of GNF:

- The grammar does not contain left recursion
- (but there can be many rules starting with the same terminal)
- *Every derivation step consumes one input symbol*

Too good to be true?

Greibach Normal Form: but why??

Useful properties of GNF:

- The grammar does not contain left recursion
- (but there can be many rules starting with the same terminal)
- *Every derivation step consumes one input symbol*

Too good to be true?

GNF grammars can be very large, i.e. $O(|G|^4)$ in general, or $O(|G|^3)$ for languages without ε.

Greibach Normal Form: but why??

Useful properties of GNF:

- The grammar does not contain left recursion
- (but there can be many rules starting with the same terminal)
- *Every derivation step consumes one input symbol*

Too good to be true?

GNF grammars can be very large, i.e. $O(|G|^4)$ in general, or $O(|G|^3)$ for languages without ε.

However, for some langauges they can result in extremely efficient parsers, because top-down parsers will only need to perform n derivation steps.

Greibach Normal Form: idea

The full algorithm is a little tedious, but the general idea is:

1. Eliminate all left recursions
2. Eliminate null productions (except on S)
3. Make substitutions to transform the grammar into the proper form:
    - Expand the first variable in each rule until we get a terminal on the left
    - Cut cycles which cannot reach a terminal

Greibach Normal Form: example

$$
\begin{aligned}
S &\to AB \mid B \\
A &\to Aa \mid b \\
B &\to Ab \mid c
\end{aligned}
$$

Eliminate left recursion

$$
\begin{aligned}
S &\to AB \mid B \\
\mathbf{A} &\to \mathbf{bR} \\
\mathbf{R} &\to \mathbf{aR} \mid \varepsilon \\
B &\to Ab \mid c
\end{aligned}
$$

Greibach Normal Form: example

$$
\begin{aligned}
S &\to AB \mid B \\
A &\to bR \\
R &\to aR \mid \varepsilon \\
B &\to Ab \mid c
\end{aligned}
$$

Eliminate null productions

$$
\begin{aligned}
S &\to AB \mid B \\
A &\to bR \mid \mathbf{b} \\
R &\to aR \mid \mathbf{a} \\
B &\to Ab \mid c
\end{aligned}
$$

Greibach Normal Form: example

$$
\begin{aligned}
S &\rightarrow AB \mid B \\
A &\rightarrow bR \mid b \\
R &\rightarrow aR \mid a \\
B &\rightarrow Ab \mid c
\end{aligned}
$$

Substitute variables on the left side of the productions of S

$$
\begin{aligned}
S &\rightarrow \mathbf{bRB} \mid \mathbf{bB} \mid \mathbf{Ab} \mid \mathbf{c} \\
A &\rightarrow bR \mid b \\
R &\rightarrow aR \mid a \\
B &\rightarrow Ab \mid c
\end{aligned}
$$

Greibach Normal Form: example

$$
\begin{aligned}
S &\to bRB \mid bB \mid Ab \mid c \\
A &\to bR \mid b \\
R &\to aR \mid a \\
B &\to Ab \mid c
\end{aligned}
$$

Substitute variables on the left side of the productions of S, again

$$
\begin{aligned}
S &\to bRB \mid bB \mid \mathbf{bRb} \mid \mathbf{bb} \mid c \\
A &\to bR \mid b \\
R &\to aR \mid a \\
B &\to Ab \mid c
\end{aligned}
$$

Greibach Normal Form: example

$$S \to bRB \mid bB \mid bRb \mid bb \mid c$$
$$A \to bR \mid b$$
$$R \to aR \mid a$$
$$B \to Ab \mid c$$

Substitute variables on the left side of the productions of B

$$S \to bRB \mid bB \mid bRb \mid bb \mid c$$
$$A \to bR \mid b$$
$$R \to aR \mid a$$
$$B \to \mathbf{bRb} \mid \mathbf{bb} \mid c$$

Greibach Normal Form: example

$$
\begin{aligned}
S &\to bRB \mid bB \mid bRb \mid bb \mid c \\
A &\to bR \mid b \\
R &\to aR \mid a \\
B &\to Ab \mid c
\end{aligned}
$$

A is unreachable now, delete it

$$
\begin{aligned}
S &\to bRB \mid bB \mid bRb \mid bb \mid c \\
R &\to aR \mid a \\
B &\to bRb \mid bb \mid c
\end{aligned}
$$

Greibach Normal Form: example

$$S \to bRB \mid bB \mid bRb \mid bb \mid c$$
$$R \to aR \mid a$$
$$B \to bRb \mid bb \mid c$$

Make a new variable N_b, to substitute in for the b's that are not at the start of a production

$$S \to bRB \mid bB \mid bR\mathbf{N_b} \mid b\mathbf{N_b} \mid c$$
$$R \to aR \mid a$$
$$B \to bR\mathbf{N_b} \mid b\mathbf{N_b} \mid c$$
$$\mathbf{N_b} \to \mathbf{b}$$

All done!

Review

- Identifying when a grammar is LL(1)
  - If left factoring is needed, then it is *not* LL(1)
  - If left recursion exists, then it is *not* LL(1)
  - It is LL(1) iff all positions in the table contain at most one production rule

Review

- Identifying when a grammar is LL(1)
  - If left factoring is needed, then it is *not* LL(1)
  - If left recursion exists, then it is *not* LL(1)
  - It is LL(1) iff all positions in the table contain at most one production rule
- Transforming grammars to LL(1)
  - Not always possible
  - Try left factoring
  - Try removing left recursion

Review

- Identifying when a grammar is LL(1)
  - If left factoring is needed, then it is *not* LL(1)
  - If left recursion exists, then it is *not* LL(1)
  - It is LL(1) iff all positions in the table contain at most one production rule
- Transforming grammars to LL(1)
  - Not always possible
  - Try left factoring
  - Try removing left recursion
- Chomsky Normal Form
  - Efficient parsing
  - Easy to build
  - All CFL have a CNF

Review

- Identifying when a grammar is LL(1)
  - If left factoring is needed, then it is *not* LL(1)
  - If left recursion exists, then it is *not* LL(1)
  - It is LL(1) iff all positions in the table contain at most one production rule
- Transforming grammars to LL(1)
  - Not always possible
  - Try left factoring
  - Try removing left recursion
- Chomsky Normal Form
  - Efficient parsing
  - Easy to build
  - All CFL have a CNF
- Greibach Normal Form
  - Derivation depth is the length of input
  - Can be excellent for parsing
  - Sometimes prohibitively large
  - All CFL have a GNF