York University
CSE 2001 – Unit 4.0 Context Free Grammars and Parsers and Context Sensitive Grammars

Instructor: Jeff Edmonds

Don't cheat by looking at these answers prematurely.

1. Consider the following grammar.

   exp ⇒ term
   ⇒ term + exp
   ⇒ term - exp
   term⇒ fact
   ⇒ fact * term
   fact ⇒ int
   ⇒ (exp)

   (a) Give a derivation/parsing tree of the following expressions. (You may use the single letters e, t, and f for exp, term, and fact.

   i. $s =$ "9+3*2".
   ii. $s =$ "(9+3)*2".
   iii. $s =$ "9-3-2".
   iv. $s =$ "(((1) * 2 + 3) * 5 * 6 + 7)".

   Answer:

```
s = 9 + 3 * 2
    |- exp -|
    t + |-e-|
    f   |-t-|
    9   f * t
        3   f
            2
s = 9 + 3 * 2
```

```
s = ( 9 + 3 ) * 2
    |--- exp ---|
    |--- term --|
    |- fact-| * t
    ( |-e-| )   f
      t + e     2
      f   t
      9   f
          3
s = ( 9 + 3 ) * 2
```

```
s = 9 - 3 - 2
    |- exp -|
    t - |-e-|
    f   t - e
    9   f   t
        3   f
            2
```

```
s = 9 - 3 - 2

s = ( ( ( 1 ) * 2 + 3 ) * 5 * 6 + 7 )
    |-exp-------------------------|
    |-term------------------------|
    |-fact------------------------|
    ( |-exp---------------------| )
      |-term-----------------| + e
      |-fact----------| * |-t-|   t
      ( |-exp-------| )   f * t   f
        |-t-----| + e     5   f   7
        |-f-| * t   t         f
        ( e )   f   f
          t     2   3
          f
          1
s = ( ( ( 1 ) * 2 + 3 ) * 5 * 6 + 7 )
```

(b) Syntactics vs Semantics

i. Syntactically, does this grammar generate reasonable expressions?

ii. Semantically (meaning), does it make any mistakes, i.e. if you evaluated the above expressions using your parsings, would you get the correct answers?

- Answer: The grammar defines reasonable syntax, but incorrect semantics. $9-3-2$ would be parsed as $9-(3-2)=9-1=8$. However, the correct semantics is $(9-3)-2=6-2=4$.

2. Recall that the regular languages are closed under complement, union, and intersection. Clearly context free languages are closed under union because if L_1 is generated with the grammar $S_1 \Rightarrow ....$ and L_2 with $S_2 \Rightarrow ....$, then $L = L_1 \cup L_2$ is generated with the grammar $S \Rightarrow S_1 \mid S_2$ together with the other rules. We will now prove that context free languages are NOT closed under intersection or under complement. Specifically, we define two languages $a^n b^n c^*$ and $a^* b^n c^n$ that are context free, however, $a^n b^n c^n = a^n b^n c^* \cap a^* b^n c^n$ is not context free. Similarly, we define language $L_?$ that is context free, however, $\overline{L_?}$ is not context free. At the end, we give a simpler example that was inspired by the fact that $\overline{L_1 \cap L_2} = \overline{L_1} \cup \overline{L_2}$ is context free but $L_1 \cap L_2$ is not.

- $L_{=*} = a^n b^n c^*$.
  Clearly this is context free.
- $L_{*=} = a^* b^n c^n$.
  Clearly this is context free.
- $L_{==} = a^n b^n c^n$.
  In class we argued that this language cannot be done by a context free grammar.
- We will prove that $L_{==} = L_{=*} \cap L_{*=}$.
  This proves that context free languages are NOT closed under intersection.

- $L_{=} = \{w_1 \# w_2 \mid w_1, w_2 \in \{0,1\}^* \text{ and } w_1 = w_2\}$.
  In class we argued that this language cannot be done by a context free grammar.
- $L_{\neq} = \{w_1 \# w_2 \mid w_1, w_2 \in \{0,1\}^* \text{ and } w_1 \neq w_2\}$.
  This question here will develop a context free grammar for this.
- $L_{\#\#} = \{w \in \{0,1,\#\}^* |$ the number of # in w is not one $\}$.
  Clearly this is context free.
- $L_? = L_{\neq} \cup L_{\#\#}$.
  Clearly this is context free.

- Note the full universe $\{0,1,\#\}^*$ is the disjoint union of $L_=$, $L_{\neq}$, and $L_{\#\#}$.
  Hence, $\overline{L_=} = L_{\neq} \cup L_{\#\#} = L_?$ and $\overline{L_?} = L_=$.
  Note $L_?$ is context free and $\overline{L_?} = L_=$ is not.
  This proves that context free languages are NOT closed under intersection.

We now fill in the details.

(a) How did we argue in class that $a^n b^n c^n$ cannot be done by a context free grammar.

- Answer: The reason is because the links cross over. Specifically, number of as and bs is linked, number of as and cs is linked, and number of bs and cs is linked. These links cross over.

(b) Argue that $a^n b^n c^n = a^n b^n c^* \cap a^* b^n c^n$.

- Answer: All the strings considered are of the form $a^\ell b^m c^n$. Being in $a^n b^n c^*$ gives that $\ell = m$. Being in $a^* b^n c^n$ gives that $m = n$. Being in both gives that $\ell = m = n$. And hence is in $a^n b^n c^n$.

(c) The build a grammar for $\{0,1\}^*$.

- Answer 1:
  $S \Rightarrow CS \mid \epsilon$
  $C \Rightarrow 0 \mid 1$
- Answer 2: I will now give the same grammar again but with different names for the non-terminals. In order to generate the languages $L_{\{0,1\}^*} = \{0,1\}^*$ and $L_{\{0,1\}} = \{0,1\}$, I will name the start non-terminal with $S_{\{0,1\}^*}$ and $S_{\{0,1\}}$.
  $S_{\{0,1\}^*} \Rightarrow S_{\{0,1\}}\ S_{\{0,1\}^*} \mid \epsilon$
  $S_{\{0,1\}} \Rightarrow 0 \mid 1$

(d) Give a grammar for $L_{=*} = a^n b^n c^*$ starting with non-terminal $S_{=*}$.

- Answer:
  $L_{a=b} = a^n b^n$
  $L_{c^*} = c^*$
  $S_{=*} \Rightarrow S_{a=b}\ S_{c^*}$
  $S_{a=b} \Rightarrow a\ S_{a=b}\ b \mid \epsilon$
  $L_{c^*} = c\ L_{c^*} \mid \epsilon$

(e) How did we argue in class that
$L_= = \{w_1 \# w_2 \mid w_1, w_2 \in \{0,1\}^* \text{ and } w_1 = w_2\}$.
cannot be done by a context free grammar.

- Answer: The reason is because the links cross over. Specifically, the first character of w_1 is linked to be the same as the first character of w_2. Similarly, the second characters and the third .... These links cross over.

(f) The build a grammar for $L_* = \{w_1 \# w_2 \mid w_1, w_2 \in \{0,1\}^*\}$ starting with the non-terminal S_*.

- Answer: $S_* \Rightarrow S_{\{0,1\}^*}\ \#\ S_{\{0,1\}^*}$

(g) We will now develop a context free grammar for
$L_{\neq} = \{w_1 \# w_2 \mid w_1, w_2 \in \{0,1\}^* \text{ and } w_1 \neq w_2\}$.
Note that if two strings w_1 and w_2 are different, then either these strings have different lengths or there is some $n \geq 0$ such that the $n{+}1^{st}$ bit of w_1 is different than that of w_2. Note we use this strange indexing because there are n bits before the $n{+}1^{st}$ bit. For example, $101\#1011 \in L_{\neq}$, because the strings 101 and 1011 have different lengths. On the other hand, $101\#111 \in L_{\neq}$, because the strings 101 and 111 are different in the 2^{nd} bit, i.e. the $n{+}1^{st}$ bit when $n = 1$.
With this motivation, define the following four languages.

- $L_{n>m} = \{w_1 \# w_2 \mid w_1, w_2 \in \{0,1\}^* \text{ and } |w_1| > |w_2|\}$.

- $L_{n<m} = \{w_1\#w_2 \mid w_1, w_2 \in \{0,1\}^* \text{ and } |w_1| < |w_2|\}$.
- $L_{0\neq 1} = \{w_1\#w_2 \mid w_1, w_2 \in \{0,1\}^*$, some $n{+}1^{st}$ bit of w_1 is 0, and of w_2 is 1 $\}$.
- $L_{1\neq 0} = \{w_1\#w_2 \mid w_1, w_2 \in \{0,1\}^*$, some $n{+}1^{st}$ bit of w_1 is 1, and of w_2 is 0 $\}$.

Suppose you already have grammars for each of these languages starting with the non-terminal named $S_{n>m}$, $S_{n<m}$, $S_{0\neq 1}$, and $S_{1\neq 0}$. Give a grammar for $L_{\neq}$ starting with the non-terminal $S_{\neq}$.

- Answer: Because we have argued that $L_{\neq} = L_{n>m} \cup L_{n<m} \cup L_{0\neq 1} \cup L_{1\neq 0}$, we can have the grammar rule
  $S_{\neq} \Rightarrow S_{n>m} \mid S_{n<m} \mid S_{0\neq 1} \mid S_{1\neq 0}$

(h) Recall that a grammar is *ambiguous* if there are more than one parsings for the same string. Argue whether or not this grammar for $L_{\neq}$ is ambiguous.

- Answer: It is ambiguous. For example, consider the string $101\#1111 \in L_{\neq}$. Because $|101| < |1111|$, we have that $101\#1111 \in L_{n<m}$. Because the second bit of 101 is a 0 and that of 1111 is a 1, we have that $101\#1111 \in L_{0\neq 1}$. Hence, there is a parsing of this string that starts with $S_{\neq} \Rightarrow S_{n>m}$ and another that starts with $S_{\neq} \Rightarrow S_{0\neq 1}$.

(i) Give grammars for the following languages:
$L_{n=m} = \{0,1\}^n\#\{0,1\}^n = \{w_1\#w_2 \mid w_1, w_2 \in \{0,1\}^* \text{ and } |w_1| = |w_2|\}$ starting with $S_{n=m}$.

- Answer: $S_{n=m} \Rightarrow S_{\{0,1\}}\ S_{n=m}\ S_{\{0,1\}} \mid \#$.

(j) $L_{n>m} = \{\{0,1\}^n\#\{0,1\}^m \mid n > m\} = \{w_1\#w_2 \mid w_1, w_2 \in \{0,1\}^* \text{ and } |w_1| > |w_2|\}$ starting with $S_{n>m}$.

- Answer: $S_{n>m} \Rightarrow S_{\{0,1\}}\ S_{\{0,1\}^*}\ S_{n=m}$.

(k) Argue that the following two languages are the same.
$L_{0\neq 1} = \{w_1\#w_2 \mid w_1, w_2 \in \{0,1\}^*$, some $n{+}1^{st}$ bit of w_1 is 0, and of w_2 is 1 $\}$
and $\{0,1\}^n0\{0,1\}^*\#\{0,1\}^n1\{0,1\}^*$.

- Answer: Fix some value of n. The first language requires that the $n{+}1^{st}$ bit of w_1 is 0. Note we use this strange indexing because there are n bits before the $n{+}1^{st}$ bit. Hence, the regular expression $\{0,1\}^n0\{0,1\}^*$ represents such w_1. Similarly for w_2.

(l) $L_{0\neq 1} = \{0,1\}^n0\{0,1\}^*\#\{0,1\}^n1\{0,1\}^*$ starting with $S_{0\neq 1}$.
Hint: Which blocks need to be linked and hence must get spewed together?

- Answer:
  Let $L_{front} = \{0,1\}^n0\{0,1\}^*\#\{0,1\}^n$
  Let $L_{plop} = 0\{0,1\}^*\#$
  Let $L_{end} = 1\{0,1\}^*$
  $S_{0\neq 1} \Rightarrow S_{front}\ S_{end}$
  $S_{front} \Rightarrow S_{\{0,1\}}\ S_{front}\ S_{\{0,1\}} \mid S_{plop}$
  $S_{plop} \Rightarrow 0\ S_{\{0,1\}^*}\ \#$
  $S_{end} \Rightarrow 1\ S_{\{0,1\}^*}$

(m) This completes the grammar for $L_{\neq}$. I find it surprising that this language has a grammar.

(n) $L_{\#\#} = \{w \in \{0,1,\#\}^* \mid$ the number of # in w is not one $\}$ starting with $S_{\#\#}$.

- Answer:
  $S_{\#\#} \Rightarrow S_{\{0,1\}^*} \mid S_{\#\geq 2}$
  $S_{\#\geq 2} \Rightarrow S_{\#=2}\ S_{\{0,1,\#\}^*}$
  $S_{\#=2} \Rightarrow S_{\{0,1\}^*}\ \#\ S_{\{0,1\}^*}\ \#\ S_{\{0,1\}^*}$
  $S_{\{0,1,\#\}^*} \Rightarrow S_{\{0,1,\#\}}\ S_{\{0,1,\#\}^*} \mid \epsilon$
  $S_{\{0,1,\#\}} \Rightarrow 0 \mid 1 \mid \#$

(o) $L_? = L_{\neq} \cup L_{\#\#}$ starting from $S_?$.

- Answer: $S_? \Rightarrow L_{\neq} \mid L_{\#\#}$

(p) Here is a second example of a language $L_?$ that is context free, however, $\overline{L_?}$ is not context free. This example was inspired by the fact that $L_1 \cap L_2$ is not context free but $\overline{L_1 \cap L_2} = \overline{L_1} \cup \overline{L_2}$ is.

- $L_{==} = \{0^\ell \# 0^m \# 0^n \mid \ell = m \text{ and } m = n\} = 0^n \# 0^n \# 0^n$.
  In class we argued that this language cannot be done by a context free grammar.
- $L_{\neq\neq} = \{0^\ell \# 0^m \# 0^n \mid \ell \neq m \text{ or } m \neq n\}$.
  Note that the complement of ($\ell = m$ and $m = n$) is (m or $m \neq n$). It is not too hard to see that this is context free.
- $L_{\#\#\#} = \{w \in \{0, \#\}^* |$ the number of # in w is not two $\}$.
  Clearly this is context free.
- $L_? = L_{\neq} \cup L_{\#\#}$.
  Clearly this is context free.
- Note the full universe $\{0, \#\}^*$ is the disjoint union of $L_{==}$, $L_{\neq\neq}$, and $L_{\#\#\#}$.
  Hence, $\overline{L_{==}} = L_{\neq\neq} \cup L_{\#\#\#} = L_?$ and $\overline{L_?} = L_{==}$.
  Note $L_?$ is context free and $\overline{L_?} = L_{==}$ is not.
  This proves that context free languages are NOT closed under intersection.

Fill in the details.

- Answer: $0^n \# 0^n \# 0^n$ is not context free for the same reason that $a^n b^n c^n$ is not.
  Here is a grammar for
  $L_{\neq\neq} = \{0^\ell \# 0^m \# 0^n \mid \ell \neq m \text{ or } m \neq n\}$
  $L_{\neq *} = \{0^\ell \# 0^m \# 0^n \mid \ell \neq m\}$
  $L_{*\neq} = \{0^\ell \# 0^m \# 0^n \mid m \neq n\}$
  $L_{\ell \neq m} = \{0^\ell \# 0^m \mid \ell \neq m\}$
  $L_{\ell < m} = \{0^\ell \# 0^m \mid \ell < m\}$
  $L_{\ell > m} = \{0^\ell \# 0^m \mid \ell > m\}$
  $L_{\ell = m} = \{0^\ell \# 0^m \mid \ell = m\}$
  $L_{0^*} = 0^*$
  $S_{\neq\neq} \Rightarrow S_{\neq *} | S_{*\neq}$
  $S_{\neq *} \Rightarrow S_{\ell \neq m} \# S_{0^*}$
  $S_{\ell \neq m} \Rightarrow S_{\ell < m} | S_{\ell > m}$
  $S_{\ell < m} \Rightarrow S_{\ell = m}\ 0\ S_{0^*}$
  $S_{\ell = m} \Rightarrow 0\ S_{\ell = m}\ 0 \mid \#$
  And so on.

3. Parsing

Look Ahead One: A grammar is said to be *look ahead one* if, given any two rules for the same non-terminal, the first place that the rules differ is a difference in a terminal. (Equivalently the rules can be views as paths down a tree.) This feature allows our parsing algorithm to look only at the next token in order to decide what to do next. Thus the algorithm runs in linear time. An example of a good set of rules would be:

A ⇒ B 'u' C 'w' E
A ⇒ B 'u' C 'x' F
A ⇒ B 'u' C
A ⇒ B 'v' G H

(Actually, even this grammar could also be problematic if when $s = bbbucccweee$, B could either be parsed as bbb or as $bbbu$. Having B *eat* the $'u'$ would be a problem.)
An example of a bad set of rules would be:

A ⇒ B C
A ⇒ D E

With such a grammar, you would not know whether to start parsing the string as a B or a D. If you made the wrong choice, you would have to back up and repeat the process.

Consider a grammar G which includes the four look ahead rules for A given above. Give the code for $GetA\,(s,i)$ that is similar to that for $GetExp\,(s,i)$. We can assume that it can be parsed, so do not bother with error detection. HINT: The code should contain NO loops.

- Answer:

  algorithm $GetA\,(s,i)$

  $\langle$***pre−cond***$\rangle$**:** s is a string of tokens and i is an index that indicates a starting point within s.

  $\langle$***post−cond***$\rangle$**:** The output consists of a parsing p of the longest substring $s[i], s[i+1], \ldots, s[j-1]$ of s that starts at index i and is a valid A. The output also includes the index j of the token that comes immediately after the parsed A .

  begin
  $\langle p_B, j_B\rangle = GetB(s,i)$
  if($s[j_B]$ = 'u')
  $\langle p_C, j_C\rangle = GetC(s, j_B+1)$
  if($s[j_C]$ = 'w')
  $\langle p_E, j_E\rangle = GetE(s, j_C+1)$
  $p_A = \langle A \ \Rightarrow \ p_B \ u \ p_C \ w \ p_E\rangle$
  $j_A = j_E$
  elseif($s[j_C]$ = 'x')
  $\langle p_F, j_F\rangle = GetF(s, j_C+1)$
  $p_A = \langle A \ \Rightarrow \ p_B \ u \ p_C \ x \ p_F\rangle$
  $j_A = j_F$
  else
  $p_A = \langle A \ \Rightarrow \ p_B \ u \ p_C\rangle$
  $j_A = j_C$
  end if
  elseif($s[j_B]$ = 'v')
  $\langle p_G, j_G\rangle = GetG(s, j_B+1)$
  $\langle p_H, j_H\rangle = GetH(s, j_G)$
  $p_A = \langle A \ \Rightarrow \ p_B \ v \ p_G \ p_H\rangle$
  $j_A = j_H$
  end if
  return $\langle p_A, j_A\rangle$
  end algorithm

4. Context Sensitive Grammar: There are no context free grammars for generating the language $\alpha\#\alpha$ where $\alpha \in \{0,1\}^*$. (The proof is an ugly pumping lemma thing, which you don't have to do.) Give a context sensitive grammar for generating this language.

   A loop invariant stating what string that grammar now has is REQUIRED. Give meaning and explanation to your non-terminals.

   The following are hints about two different ways to do it. Think about both, but only hand in one.

   (a) Start with $S \to S'<$
   $S' \to 0S'0 \mid 1S'1 \mid \#>Q$
   This produces the string $\alpha\#>Q\alpha^R<$
   Then move the head Q back and forth to reverse the order of α^R giving α.

   - Answer:
     Loop Invariant: Let α_i be the first i bits of α and β_i be the rest so that $\alpha = \alpha_i\beta_i$.
     The string that the grammar has now is $\alpha\#\alpha_i>Q\beta_i^R<$.
     If there are no more characters stop.
     $>Q< \to \epsilon$

Otherwise, move the Q right to the $<$.

$Q0 \to 0Q$

$Q1 \to 1Q$

Remember and delete last the character and get ready to move left.

$0Q< \to L_0<$

$1Q< \to L_1<$

Move left to the $>$.

$0L_0 \to L_00$

$1L_0 \to L_01$

$0L_1 \to L_10$

$1L_1 \to L_11$

Put the remembered character in place and complete the loop invariant.

$>L_0 \to 0>Q$

$>L_1 \to 1>Q$

(b) Start with $S \to S'<$

$S' \to 0S'C_0 \mid 1S'C_1 \mid \#$

This produces the string $\alpha\#\beta<$

where $\beta = \alpha^R$, except 0 is replaced by C_0 and 1 by C_1.

Then have the C_0 and the C_1 move on their own and convert when in place to 0 and 1.

- Answer:

  Loop Invariant: Let α_i be the first i bits of α and β_i be the rest so that $\alpha = \alpha_i\beta_i$.

  In the string that the grammar has now, the α_i part that had been C_0/C_1 has been converted to 0/1 and reversed so that it is is in the same direction as the left α. The β_i is still in C_0/C_1 and not yet reversed. However, these two strings are intertwined in arbitrary ways.

  Let the 0/1 characters stay in the same order relative to each other but move left of the C_0/C_1.

  $\forall c, c'$, there is the rule $C_cc' \to c'C_c$

  The C_0/C_1 at the far right can covert to 0/1.

  $\forall c$, there is the rule $C_c< \to c<$

  At any point the end $<$ can be removed, but it better not happen until all the C_0/C_1 have been converted or else the string is not actually generated.

  $< \to \epsilon$