CS273 Spring 2006
HW6 solutions

Problem 1

Give context-free grammars for the following languages:

1. $\{a^i b^j : 0 \le i \le j \le 2i\}$
2. Strings with characters '$<$', '$>$', '(', and ')' that are balanced with respect to both types of parentheses. This means that 2 matched pairs of parentheses are either disjoint or have one nested within the other. Thus, "$(<>)<>$" is a valid string, but "$(<)>$" is not since the two pairs intersect without being nested.

Solution: In the solution below, the nonterminal variables use **bold** fonts, and terminals use regular fonts. (1)

$$\mathbf{S} \to \epsilon \mid a\mathbf{S}b \mid a\mathbf{S}bb$$

(2)

$$\mathbf{S} \to \epsilon \mid (\mathbf{S}) \mid < \mathbf{S} > \mid \mathbf{SS}$$

Problem 2

You have been asked to write a grammar for the new programming language CPL (for Cool Programming Language). Programs in CPL use strings of lower-case letters for variable names and may include natural numbers as constants. It has five types of statements. The first are assignment statements, where a sum of variables and constants is assigned to a variable. For example,

```
newx := oldx + y + 3
```

assigns the value of the expression on the right to the variable `x`. The second type of statement is the `if` statement, where a condition is tested to decide which of two lists of statements to execute. For example,

```
if( x < z + 18 ) then { x := 4 }
                 else { x := 17 + 5
                        y := x + y }
```

tests the condition `x < z + 18` and executes one of the two lists of statements in braces. A condition can compare any two sums of variables and constants with one of the operators '$<$', '$>$', or '='. If statements must include both 'then' and 'else' clauses, though either list of statements can be empty. The third type of statement are `while` loops, where a list of statements is executed until a condition fails, as in

```
while( x + 5 > y + 24 ) {
  x := x + 1
  y := y + y
}
```

The last two types of statements perform input and output. Input is performed with the statement

```
read(v)
```

where the argument to `read` can be any variable. Output is performed with the statement

```
write(x+9+y)
```

where the argument to `write` is any sum of variables and constants.

Give a context-free grammar for programs in CPL. Your grammar should include the underscore character '_' wherever whitespace (spaces, tabs, and newlines) is allowed. This character is assumed to represent any amount of whitespace. You can also use "[a–z]" as shorthand to denote a single alphabetic character to save yourself writing out a separate production for each character.

Solution: In the solution below, we use $\Box$ to represent any number of whitespaces; and use $\Diamond$ to represent at least one whitespaces. The nonterminal variables use **bold** fonts, and terminals use regular fonts.

$$
\begin{aligned}
\mathbf{S} &\rightarrow \Box\mathbf{stats}\Box \\
\mathbf{stats} &\rightarrow \mathbf{stat} \mid \mathbf{stat}\Diamond\mathbf{stats} \mid \epsilon \\
\mathbf{stat} &\rightarrow \mathbf{stat1} \mid \mathbf{stat2} \mid \mathbf{stat3} \;\; \mid \mathbf{stat4} \mid \mathbf{stat5} \\
\mathbf{stat1} &\rightarrow \mathbf{var}\Box{:=}\Box\mathbf{expr} \\
\mathbf{stat2} &\rightarrow \text{if}\Box(\Box\mathbf{boolexpr}\Box)\Box\text{then}\Box\{\Box\mathbf{stats}\Box\}\Box\text{else}\Box\{\Box\mathbf{stats}\Box\} \\
\mathbf{stat3} &\rightarrow \text{while}\Box(\Box\mathbf{boolexpr}\Box)\Box\{\Box\mathbf{stats}\Box\} \\
\mathbf{stat4} &\rightarrow \text{read}\Box(\Box\mathbf{var}\Box) \\
\mathbf{stat5} &\rightarrow \text{write}\Box(\Box\mathbf{expr}\Box) \\
\mathbf{expr} &\rightarrow \mathbf{expr}\Box\mathbf{op}\Box\mathbf{expr} \mid \mathbf{var} \mid \mathbf{const} \\
\mathbf{boolexpr} &\rightarrow \mathbf{expr}\Box\mathbf{boolop}\Box\mathbf{expr} \\
\mathbf{boolop} &\rightarrow = \mid > \mid < \\
\mathbf{op} &\rightarrow + \mid - \mid * \mid / \\
\mathbf{const} &\rightarrow [\text{1-9}]\ \mathbf{consthelper} \mid 0 \\
\mathbf{consthelper} &\rightarrow [\text{0-9}]\ \mathbf{consthelper} \mid \epsilon \\
\mathbf{var} &\rightarrow \mathbf{var}\ \mathbf{var} \mid [a-z]
\end{aligned}
$$

Comments: There are many solutions to this problem. For example, some students defined the binary operator as + only. This is certainly acceptable, since the question specified + only. We generally accept any solution that makes sense.

Problem 3

Consider the context-free grammar G defined by productions

$$S \rightarrow aS|Sb|a|b$$

1. Prove by induction on the string length that no string in $L(G)$ has ba as a substring.
2. Prove that $L(G)$ is strings where all occurrences of 'a' come before any occurrences of 'b'. (Recall that this requires showing that $L(G)$ contains all such strings and that all such strings are in $L(G)$.)

Solution:

1. We prove the following claim.

   Claim 1 *For any string $\beta \in L(G)$, it does not contain* ba *as a substring.*

   Proof: We use induction on the length of β. The base case of $|\beta| = 1$ is clearly true. Suppose the claim holds when $|\beta| = n$, we next show the claim holds when $|\beta| = n + 1$.

   Consider the process of deriving β by applying productions from head to body. There are two possibilities:

(a) $S \Rightarrow aS \Rightarrow^* \beta$. In this case, $\beta = a\beta'$, where β' is a string of $L(G)$ and $|\beta'| = |\beta| - |a| = n$. By induction assumption, β' does not contain ba as a substring. Therefore, $a\beta'$ does not contain ba as a substring either.

(b) $S \Rightarrow Sb \Rightarrow^* \beta$. In this case, $\beta = \beta' b$, where β' is a string of $L(G)$ and $|\beta'| = |\beta| - |b| = n$. By induction assumption, β' does not contain ba as a substring. Therefore, $\beta' b$ does not contain ba as a substring either.

■

2. We are concerned with the set of strings defined over 'a' and 'b'. Let L_1 be the set of strings that contains ba as a substring; and let L_2 be the set of strings where all occurrences of 'a' come before any occurrence of 'b'. It is easy to see that $L_2 = \overline{L_1}$.

   The question asks us to prove $L(G) = L_2$. We do this by proving $L(G) \subseteq L_2$ and $L_2 \subseteq L(G)$.

   (a) The part (1) implies that $L(G) \subseteq \overline{L_1} = L_2$.

   (b) To prove $L_2 \subseteq L(G)$, we need to show that for any string $\beta \in L_2$ (that is, β has all occurrences of 'a' come before any occurrence of 'b'), it holds that $\beta \in L(G)$.

   We prove this by induction on the length of β. The base case of $|\beta| = 1$ is clearly true. Suppose the claim holds when $|\beta| = n$, we next show the claim holds when $|\beta| = n + 1$. Suppose $\beta = a^i b^{n+1-i}$, where $0 \le i \le n + 1$. If $i = 0$, then $\beta = b^{n+1}$ can be derived by $S \Rightarrow Sb \Rightarrow^* b^n b$ (the second derivation $S \Rightarrow^* b^n$ is because of the induction assumption). Otherwise, we have $S \Rightarrow aS \Rightarrow^* aa^{i-1}b^{n+1-i}$ (the second derivation $S \Rightarrow^* a^{i-1}b^{n+1-i}$ is because of the induction assumption).

Comments: Some students used induction to prove 2(a) directly, which is correct; however, they apparantly did not realize that they could use part (1). Some students did not realize that they need to prove 2(b).

Problem 4

Consider the CFG G defined by productions

$$S \to aSbS|bSaS|\epsilon$$

Prove that $L(G)$ is the set of all strings with equal numbers of occurrences of 'a' and 'b'. (Again, remember to show both directions.)

Solution: We are concerned with the strings defined over 'a's and 'b's. Let L be the set of strings with equal numbers of occurrences of 'a' and 'b'. We show $L(G) = L$ by proving $L(G) \subseteq L$ and $L \subseteq L(G)$.

In the following, for a string β, we denote by $\#^a(\beta)$ the number of 'a's in β, and denote by $\#^b(\beta)$ the number of 'b's in β.

1. **Claim 2** *$L(G) \subseteq L$. That is, for any string $\beta \in L(G)$, we have $\#^a(\beta) = \#^b(\beta)$.*

   Proof: Usually, there may exist many ways to derive β. Define $g(\beta)$ to be a derivation of β with the minimal derivation length, and define $|g(\beta)|$ to the length of the derivation $g(\beta)$.

   We use induction on $|g(\beta)|$. The base case $|g(\beta)| = 1$ is clearly true (which correponds to $\beta = \epsilon$). We assume that when $|g(\beta)| < n$, the claim holds. Next we show that the claim holds when $|g(\beta) = n|$.

   Consider how β is derived by applying productions from head to body in the derivation $g(\beta)$. There are two possibilities.

(a) $S \Rightarrow aSbS \Rightarrow^* \beta$. In this case, $\beta = a\beta' b\beta''$, where β' and β'' are strings of $L(G)$. In addition, $|g(\beta')| \leq |g(\beta)| - 1 = n - 1$, and $|g(\beta'')| \leq |g(\beta)| - 1 = n - 1$. By induction assumption, $\#^a(\beta') = \#^b(\beta')$ and $\#^a(\beta'') = \#^b(\beta'')$. Now it is easy to verify that β contains equal number of occurrences of 'a's and 'b's. Indeed, $\#^a(\beta) = 1 + \#^a(\beta') + \#^a(\beta'') = 1 + \#^b(\beta') + \#^b(\beta'') = \#^b(\beta)$.

(b) $S \Rightarrow bSaS \Rightarrow^* \beta$. We can prove β contains equal number of occurrences of 'a' and 'b', by similar arguments as used above.

■

2. **Claim 3** $L \subseteq L(G)$. *That is, for any string β with $\#^a(\beta) = \#^b(\beta)$, we have $\beta \in L(G)$.*

   Proof: First, note that β must have even length since it consists of equal number of 'a's and 'b's.

   Again, we use induction on the length of β. The base case $|\beta| = 0$ is clearly true. We assume that when $|\beta| < n$, the claim holds. Next we show that the claim holds when $|\beta| = n$.

   We denote the ith character of β by $\beta[i]$. For example, if $\beta = aababb$, then $\beta[1] =$'a' and the last character of β is $\beta[6] =$'b'. We denote the substring of β from index i to j by $\beta[i \dots j]$.

   Without loss of generality, we assume that β begins with 'a'; that is, $\beta[1] =$ 'a'. We are interested in all 'b's in β. Let $b_1, \dots, b_{n/2}$ be the indexes of all 'b's in β (recall that n must be an even number). Define $f(b_i)$ be the difference between the number of 'b's and the number of 'a's in $\beta[1 \dots b_i]$. It is easy to see that $f(b_1) \leq 0$, since there are at least one 'a' before the first 'b'; and $f(b_{n/2}) \geq 0$, since there are at most $n/2$ 'a's before the last 'b'.

   Suppose there exists an index b_i such that $f(b_i) = 0$. In other words, both the substring $\beta[2 \dots b_i - 1]$ (this may be an empty string) and the substring $\beta[b_i + 1 \dots n]$ (this may also be an empty string) have equal number of 'a' and 'b's. By induction assumption, $S \Rightarrow^* \beta[2 \dots b_i - 1]$ and $S \Rightarrow^* \beta[b_i + 1 \dots n]$. Therefore, $S \Rightarrow aSbS \Rightarrow^* a\beta[2 \dots b_i - 1]b\beta[b_i + 1 \dots n]$, which is exactly β.

   Therefore, it suffices to prove that there exists an index b_i such that $f(b_i) = 0$. If either $f(b_1) = 0$ or $f(b_{n/2}) = 0$, we are done. Otherwise, we must have $f(b_1) < 0$ and $f(b_{n/2}) > 0$. Now observe that $f(b_{i+1}) \leq f(b_i) + 1$. This implies that there must be some i, such that $1 < i < n/2$ and $f(b_i) = 0$. ■

Comments: Less than a quarter of students got this problem completely right. Most students could not prove part 2. Some students seemed to think $\mathbf{S} \rightarrow a\mathbf{S}b\mathbf{S}$ implies that the two **S**s on the right side should derive exactly the same strings.