Problem 1

To design the grammar for the functions and arrays, it is easiest to do it in a top down fashion. The “outer-most” construct is either a function call or an array lookup. If $S$ is the start variable, then we have the production rule $S \rightarrow F \mid A$ where $F$ is a function call and $A$ is an array lookup. The name of a functions is either $f$ or $g$ and it is followed by an argument list enclosed in parentheses given $F \rightarrow f(L) \mid g(L)$. If $R$ is an argument, then the argument list is either empty or contains one or more arguments.

An array lookup is an array name, either $a$ or $b$ followed by one or more $[R]$ where $R$ is again an argument. The full grammar is given in Figure 1.

Problem 2

Recall that a regular expression is built inductively according to the following 6 rules.

1. $\emptyset$
2. $\varepsilon$
3. $\sigma$
4. $R_1 R_2$
5. $R_1 + R_2$
6. $R_1^*$
the regular expression we wish to convert to a CFG. For cases 4–6, we assume that we have grammars

\[ G \]

in nation of two smaller regular expressions, the union of two smaller regular expressions, or the Kleene star of

\[ \sigma \]

That is, any regular expression is the empty set, the empty string, any alphabet symbol \( \sigma \in \Sigma \), the concatenation of two smaller regular expressions, the union of two smaller regular expressions, or the Kleene star of a smaller regular expression.

To convert from a regular expression to a CFG, we need only give a construction for each of the six cases. By structural induction, this gives a complete construction for any regular expression. Let \( R \)

\[ \sigma \]

be any regular expression. The first step to this is coming up with the complement of \( L \).

To apply this construction to \( (a(b+a)^*+bb)\ast(a(b+ba)^* \)

is how the construction works.

That is, any regular expression is the empty set, the empty string, any alphabet symbol \( \sigma \in \Sigma \), the concatenation of two smaller regular expressions, the union of two smaller regular expressions, or the Kleene star of a smaller regular expression.

1. If \( R = \emptyset \), then let \( G = (\{ S \}, \Sigma, \emptyset, S) \)—that is, the grammar with one variable and no production rules. Clearly, \( L(G) = \emptyset \).
2. If \( R = \varepsilon \), then let \( G = (\{ S \}, \Sigma, \{ S \to \varepsilon \}, S) \) which generates only the empty string.
3. If \( R = \sigma \) for some \( \sigma \in \Sigma \), then similarly to the previous case, we have \( G = (\{ S \}, \Sigma, \{ S \to \sigma \}, S) \).
4. If \( R = R_1R_2 \), then we let \( V = V_1 \cup V_2 \cup \{ S \} \) and \( P = P_1 \cup P_2 \cup \{ S \to S_1S_2 \} \).
5. If \( R = R_1 + R_2 \), then we let \( V = V_1 \cup V_2 \cup \{ S \} \) and \( P = P_1 \cup P_2 \cup \{ S \to S_1 | S_2 \} \).
6. If \( R = R_1^* \), then we let \( V = V_1 \cup \{ S \} \) and \( P = P_1 \cup \{ S \to \varepsilon | S_1S \} \).

To apply this construction to \( (a(b+a)^*+bb)\ast(a(b+ba)^* \)

we want \( L(A) = L((a(b+a)^*+bb)\ast \) and \( L(B) = L(a(b+ba)^* \) \). We next want to apply rule 6 to \( A \) to get \( A \to \varepsilon | CA \) where \( L(C) = L(a(b+ba)^* + bb) \). Continuing in this depth-first fashion we end up with the grammar in Figure 2. Note that many of the variables are redundant, for example \( F \) and \( J \). However, this is how the construction works.

\[
\begin{align*}
S &\to AB \\
A &\to \varepsilon \mid CA \\
C &\to D \mid E \\
D &\to FG \\
F &\to a \\
G &\to \varepsilon \mid HG \\
H &\to I \mid J \\
J &\to a \\
E &\to KL \\
K &\to b
\end{align*}
\]

\[ L \to b \]

\[ B \to MN \]

\[ M \to a \]

\[ N \to \varepsilon \mid ON \]

\[ O \to P \mid Q \]

\[ P \to RT \]

\[ R \to a \]

\[ T \to b \]

\[ Q \to UV \]

\[ U \to b \]

\[ V \to a \]

Figure 2: CFG for the regular expression \( (a(b+a)^*+bb)\ast(a(b+ba)^* \)

**Problem 3**

The first step to this is coming up with the complement of \( L = \{ a^kb^n a^m \mid 0 \leq k \leq n < m \} \). First note that if \( w \in L^c \), the complement of \( L \), then \( w \) must be either empty or contain at least one \( b \). If \( w \in L^c \) is of the
form $a^kb^n a^m$—note that we must have $n > 0$—then we must have either $k > n$ or $n \geq m$. If $w \in L^c$ is not of the form $a^kb^n a^m$, then $w$ must contain $baa^*b$ since it must contain at least one $b$. Therefore,

$$L^c = \{a^kb^n a^* \mid k > n > 0\} \cup \{a^* b^n a^m \mid n \geq m, \ n > 0\} \cup \{\varepsilon\} \cup \{\Sigma^*baa^*b\Sigma^*\}.$$ 

Thus, $L^c$ is the union of 4 context free languages. Ignoring $\varepsilon$ for the moment, we can build PDAs for the three other constituent parts of $L^c$ and then connect them via $\varepsilon$ transitions from the start state. By making the start state an accept state, we can accept $\varepsilon$. Figure 3 contains the full PDA for $L^c$. 

Figure 3: PDA for $L^c$. 

3