CS 334 – Fall 2004: Answers to Assignment 7

1. page 204: 5.3.2

```
B -> BB | (B) | [B] | Λ
```

2. page 205: 5.3.4a

Change production (5) to:

```
    ListItem -> <LI> Doc </LI>
```

3. page 214: 5.4.1

Here are the parse trees:

```
           S                   S
          / |               / / | \
         a  S              a S  b  S
          / | \ \            | \    |
         a  S  b S           a  S  e
            |    |              |
            e    e              e
```

The two leftmost derivations are: S => aS => aaSbS => aabS => aab and S => aSbS => aaSbS => aabS => aab.

The two rightmost derivations are: S => aS => aaSbS => aaSb => aab and S => aSbS => aSb => aaSb => aab.

4. page 214: 5.4.3

The idea is to introduce another nonterminal *T* that cannot generate an unbalanced *a*. That strategy corresponds to the usual rule in programming languages that an ``else" is associated with the closest previous, unmatched ``then." Here, we force a *b* to match the previous unmatched *a*. The grammar:

```
    S -> aS | aTbS | Λ
    T -> aTbT | Λ
```

5. page 214: 5.4.6

It is not. We need to have three nonterminals, corresponding to the three possible ``strengths" of expressions:

1. A *factor* cannot be broken by any operator. These are the basis expressions, parenthesized expressions, and these expressions followed by one or more *'s.
2. A *term* can be broken only by a *. For example, consider 01, where the 0 and 1 are concatenated, but if we follow it by a *, it becomes 0(1*), and the concatenation has been ``broken" by the *.
3. An *expression* can be broken by concatenation or *, but not by +. An example is the expression 0+1. Note that if we concatenate (say) 1 or follow by a *, we parse the expression 0+(11) or 0+(1*), and in either case the union has been broken.

The grammar:

```
E -> E+T | T
T -> TF | F
F -> F* | (E) | 0 | 1 | ∅ | Λ
```

6. page 245: 6.3.1

({q},{0,1),{0,1,A,S},δ,q,S) where delta is defined by:

1. δ(q,e,S) = {(q,0S1), (q,A)}
2. δ(q,e,A) = {(q,1A0), (q,S), (q,Λ)}
3. δ(q,0,0) = {(q,Λ)}
4. δ(q,1,1) = {(q,Λ)}

7. page 246: 6.3.3

1. S -> [qZq] | [qZp]

The following four productions come from rule (1).

2. [qZq] -> 1[qXq][qZq]
3. [qZq] -> 1[qXp][pZq]
4. [qZp] -> 1[qXq][qZp]
5. [qZp] -> 1[qXp][pZp]

The following four productions come from rule (2).

6. [qXq] -> 1[qXq][qXq]
7. [qXq] -> 1[qXp][pXq]
8. [qXp] -> 1[qXq][qXp]
9. [qXp] -> 1[qXp][pXp]

The following two productions come from rule (3).

10. [qXq] -> 0[pXq]
11. [qXp] -> 0[pXp]

The following production comes from rule (4).

12. [qXq] -> Λ

The following production comes from rule (5).

13. [pXp] -> 1

The following two productions come from rule (6).

14. [pZq] -> 0[qZq]
15. [pZp] -> 0[qZp]

8. page 251: 6.4.1b

Not a DPDA. For example, rules (3) and (4) give a choice, when in state *q*, with 1 as the next input symbol, and with *X* on top of the stack, of either using the 1 (making no other change) or making a move on epsilon input that pops the stack and going to state *p*.

9. page 251: 6.4.3a

Suppose a DPDA *P* accepts both *w* and *wx* by empty stack, where *x* is not Λ (i.e., N(P) does not have the prefix property). Then (q_0,wxZ_0) |-* (q,x,Λ) for some state *q*, where q_0 and Z_0 are the start state and symbol of *P*. It is not possible that (q,x,Λ) |-* (p,Λ,Λ) for some state *p*, because we know *x* is not Λ, and a PDA cannot have a move with an empty stack. This observation contradicts the assumption that *wx* is in *N(P)*.