ECS 140A HW 1

Yiming Peng

1. (a) aabccd

    No, the string cannot be derived from the grammar.

    <S> ::= a <S> c <B>

    ::= a a <S> c <B> c <B>

    ::= a a b c <B> c <B>

    We can not get c after c from <B> in this case, so it is not possible to get string aabccd from the grammar.

   (b) accbcc

    No, the string cannot be derived from the grammar.

    <S> ::= a <S> c <B>

    ::= a <A> c <B>

    ::= a c c <B>

    We can not get b from <B>, only d or <A> can be replaced by <B>. Thus, it's impossible to get string accbcc from the grammar.

   (c) accccc

   Yes, this string can be derived from the grammar.

   <S> ::= a <S> c <B>

   ::= a <A> c <B>
   ::= a c <A> c <B>
   ::= a c c <A> c <B>
   ::= a c c c c <B>
   ::= a c c c c <A>
   ::= a c c c c c

2. (a) Convert the grammar into EBNF.

    <integer> ::= [<sign>] <unsigned>
    <unsigned> ::= {<digits>} <digits>
    <digits> ::= <digit> {<digit>}
    <digit> ::= 0 | 1 | 2 | 3| 4 | 5 | 6 | 7 | 8 | 9
    <sign> ::= + | -

3. (a) <S> ::= ab | a <S> b

<S> ::= ab

<S> ::= a<S>b = a a b b

<S> ::= a<S>b = a a <S> b b = a a a b b b

Thus, this BNF can generate a^nb^n where n > 0. The language consists of the strings that contain n letter a followed by n letter b.

(b) <S> ::= <A> <B> <C>
<A> ::= a <A> | a
<B> ::= b <B> | b
<C> ::= c <C> | c

<S> ::= a b c = a <A> b <B> c <C> = many possible solutions

Thus, this BNF can generate $a^m b^n c^k$ where m, n, k > 0. The language has strings that contain one or more a's followed by one or more b's followed by one or more c's.

(c) <S> ::= <x> | <y>
<x> ::= 0 <x> 1 | <x1>
<x1> ::= 0 <x1> | 0
<y> ::= 0 <y> 1 1 | <y1>
<y1> ::= <y1> 1 | 1

when<S> ::= <x>,

<S> ::= <x> | <y> = <x> = <x1> = 0

<S> ::= <x> | <y> = <x> = <x1> = 0 <x1> = 0 0

…

<S> ::= <x> | <y> = <x> = 0 <x> 1 = 0 <x1> 1 = 0 0 1

<S> ::= <x> | <y> = <x> = 0 <x> 1 = 0 <x1> 1 = 0 0 <x1> 1 = 0 0 0 1

…

<S> ::= <x> | <y> = <x> = 0 <x> 1 = 0 0 <x> 1 1 = 0 0 <x1> 1 1 = 0 0 0 1 1

<S> ::= <x> | <y> = <x> = 0 <x> 1 = 0 0 0 <x> 1 1 1 = 0 0 0 <x1> 1 1 1 = 0 0 0 0 1 1 1

…

Thus, the string will be at least n+1 0's and n 1's, which is: 0^n1^m, where $n \in N$, $m \in N$, and $n >= m+1$.

when<S> ::= <y>,

<S> ::= <x> | <y> = <y> = <y1> = 1

<S> ::= <x> | <y> = <y> = <y1> = <y1> 1 = 1 1

…

<S> ::= <x> | <y> = <y> = 0 <y> 1 1 = 0 <y1> 1 1 = 0 1 1 1

<S> ::= <x> | <y> = <y> = 0 <y> 1 1 = 0 <y1> 1 1 = 0 <y1> 1 1 1 = 0 1 1 1 1

…

<S> ::= <x> | <y> = <y> = 0 <y> 1 1 = 0 0 <y> 1 1 1 1 = 0 0 <y1> 1 1 1 1 = 0 0 1 1 1 1 1

<S> ::= <x> | <y> = <y> = 0 <y> 1 1 = 0 0 <y> 1 1 1 1 = 0 0 0 <y1> 1 1 1 1 1

= 0 0 0 1 1 1 1 1 1 1

…

Thus, the string will be n 0's followed by at least 2n+1 1's, which is: 0^n1^m, where $n \in N$, $m \in N$, and $m >= 2n+1$.

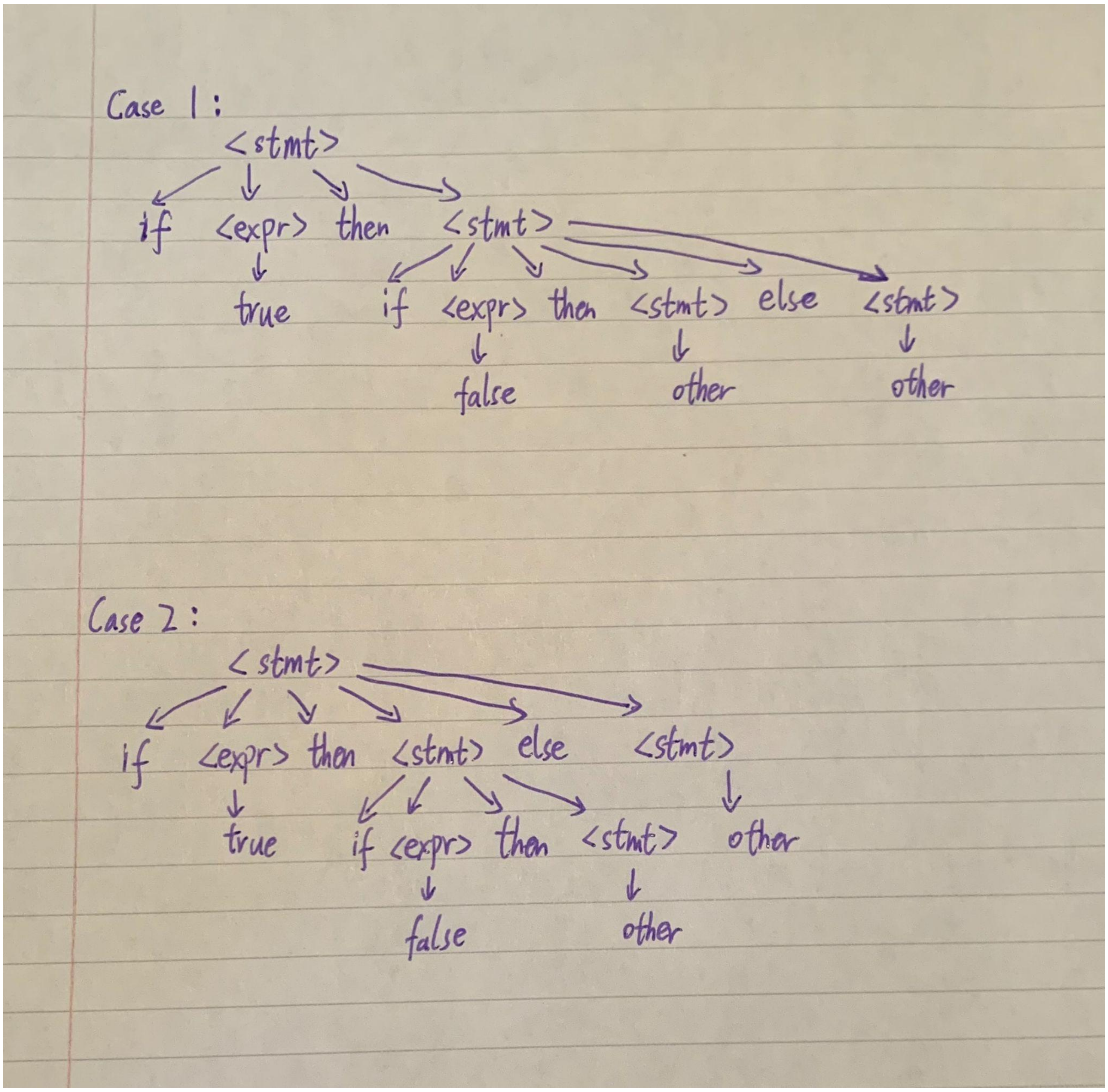
In conclusion, the string generated by this grammar is:

{0^n1^m, where $n \in N$, $m \in N$, and $n >= m+1$} Union {0^n1^m, where $n \in N$, $m \in N$, and $m >= 2n+1$.}

4. (a) This grammar is ambiguous. Give a string having two different parse trees and

draw the parse trees.

Ambiguous string sample: If true then if false then other else other



(b) If we adopt the disambiguating rule (used in most languages) match each else with the closest previous unmatched then, write an equivalent, unambiguous grammar.

<stmt> ::= <temp> | if <expr> then <stmt>
| if <expr> then <temp> else <stmt>
<temp> ::= if <expr> then <temp> else <temp> | other

<expr> ::= true | false

5. (a) The set of all strings consisting of zero or more a's.

BNF <S> ::= a <S> | <empty>

EBNF <S> ::= {a}

(b) The set of all strings consisting of one or more a's, where there is a comma in between each a and the following a. Note that there is no comma before the first a or after the last a.

BNF <S> ::= a , <S> | a

EBNF <S> ::= a {,a}