CS143 Midterm Sample Solution
Fall 2010

- Please read all instructions (including these) carefully.
- There are 4 questions on the exam, all with multiple parts. You have 75 minutes to work on the exam.
- The exam is closed book, but you may refer to your four sheets of prepared notes.
- Please write your answers in the space provided on the exam, and clearly mark your solutions. You may use the backs of the exam pages as scratch paper. Please do not use any additional scratch paper.
- Solutions will be graded on correctness and clarity. Each problem has a relatively simple and straightforward solution. You may get as few as 0 points for a question if your solution is far more complicated than necessary. Partial solutions will be graded for partial credit.

NAME: _____

In accordance with both the letter and spirit of the Honor Code, I have neither given nor received assistance on this examination.

SIGNATURE: _____

Problem	Max points	Points
1	15	
2	20	
3	20	
4	20	
TOTAL	75	

1. **First and Follow Sets** (15 points)
Consider the following grammar with 2 missing productions:

$$\begin{aligned} S &\rightarrow aS \mid \ldots \textbf{(1)} \\ A &\rightarrow \ldots \textbf{(2)} \mid \epsilon \\ X &\rightarrow cS \mid \epsilon \\ Y &\rightarrow dS \mid \epsilon \\ Z &\rightarrow eS \end{aligned}$$

Fortunately we have the First and Follow sets for this grammar:

	First	Follow
S	$\{a, b, c, d, e\}$	$\{\$\} \cup Follow(X) \cup Follow(Y) \cup Follow(Z)$
A	$\{c, d, e, \epsilon\}$	$\{b\}$
X	$\{c, \epsilon\}$	$First(Y)/\epsilon \cup First(Z)$
Y	$\{d, \epsilon\}$	$First(Z)$
Z	$\{e\}$	$Follow(A)$
a	$\{a\}$	$First(S)$
b	$\{b\}$	$Follow(S)$
c	$\{c\}$	$First(S)$
d	$\{d\}$	$First(S)$
e	$\{e\}$	$First(S)$

Reconstruct the grammar by filling in the missing productions. Note that you can only define two productions. (And recall that $X \rightarrow A|B$ is two productions).

Answer:
(1) Ab
(2) XYZ

Since First(A) has c,d,e, **(2)** cannot start with a terminal, so we have to use X, Y, and Z to generate c,d,e. Since Z does not go to ϵ and Follow(A) $\subseteq$ Follow(Z), we can conclude that Z has to go last. Since First(Y) $\subseteq$ Follow(X) and First(Z) $\subseteq$ Follow(Y), we can conclude that the order has to be XYZ.

First(S) has a,b,c,d,e, so **(1)** should be able to generate b,c,d,e in the first position. Since we have no rule that produces b, it needs to have b, but the b cannot be the first position, since that wouldn't allow another character to be in the first position. Follow(S) $\subseteq$ Follow(b), so b would be the last character in this production. To generate c,d,e in the first position we have to use A, since it also has to be able to go to ϵ. Follow(A) = b, so we know there are no other (non-)terminals in between the A and b. (Something like Abbbb would also work, though we didn't directly specify b $\in$ Follow(b))

2. **Semantic Actions** (20 points)
Consider the following grammar for generating binary fractions.

$$\begin{aligned} F &\rightarrow 0.B \\ B &\rightarrow 0B \mid 1B \mid 0 \mid 1 \end{aligned}$$

- Fill in the missing semantic actions below to calculate the decimal value of an input string. Recall that the numeric value of a binary fraction $0.b_1b_2 \ldots b_n$ is calculated as $\Sigma_{i=1}^{n}\ b_i 2^{-i}$. Each non-terminal has a synthesized attribute val that is used to store its value. The final value should be returned in $F.val$.

$F \rightarrow 0.B$ { $F.val = B.val$ }

$B_0 \rightarrow 0B_1$ {

}

$B_0 \rightarrow 1B_1$ {

}

$B \rightarrow 0$ {

}

$B \rightarrow 1$ {

}

Answer:

$$\begin{aligned} F &\rightarrow 0.B\ \{F.val = B.val\} \\ B_0 &\rightarrow 0B_1\ \{B_0.val = B_1.val/2\} \\ B_0 &\rightarrow 1B_1\ \{B_0.val = B_1.val/2 + 1/2\} \end{aligned}$$

$$
\begin{array}{lcll}
B & \rightarrow & 0 & \{B.val = 0\} \\
B & \rightarrow & 1 & \{B.val = 1/2\}
\end{array}
$$

- Consider a grammar for binary strings or their complements:

$$\begin{aligned} N &\rightarrow B \mid \neg B \\ B &\rightarrow B0 \mid B1 \mid 0 \mid 1 \end{aligned}$$

The value of a (non-negated) string is just the decimal value of the binary number the string represents; the value of a negated string is the decimal value of the string with 1's replaced by 0's and 0's replaced by 1's. For example, the value of 010 is 2 and ¬010 is 5. Fill in the semantic actions below to calculate the decimal value of an input string. Each non-terminal has a synthesized attribute *val* that is used to store its value, with the final value in $N.val$. Your solution must use at least one inherited attribute.

$N \rightarrow B$ {

}

$N \rightarrow \neg B$ {

}

$B_0 \rightarrow B_1 0$ {

}

$B_0 \rightarrow B_1 1$ {

}

$B \rightarrow 0$ {

}

$B \rightarrow 1$ {

}

Answer:

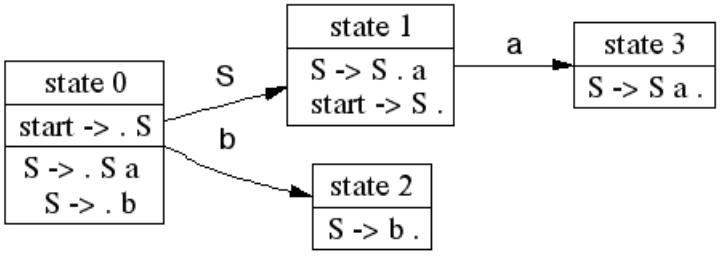
$$
\begin{array}{rcll}
N & \rightarrow & B & \{N.val = B.val; B.neg = false\} \\
N & \rightarrow & \neg B & \{N.val = B.val; B.neg = true\} \\
B_0 & \rightarrow & B_1 0 & \{B_0.val = 2 * B_1 + (B_0.neg)?1 : 0;\ B_1.neg = B_0.neg\} \\
B_0 & \rightarrow & B_1 1 & \{B_0.val = 2 * B_1 + (B_0.neg)?0 : 1;\ B_1.neg = B_0.neg\} \\
B & \rightarrow & 0 & \{B.val = (B.neg)?1 : 0\} \\
B & \rightarrow & 1 & \{B.val = (B.neg)?0 : 1\}
\end{array}
$$

3. **Parsing Algorithms** (20 points)
Give a grammar that is SLR(1) but cannot be recognized by a recursive descent parser. Your grammar must have the following properties:

- It should contain no more than two productions (again, remember $X \rightarrow A|B$ is two productions).
- The language generated by the grammar should contain at least two distinct strings.
- The grammar should be as simple as possible, given the preceding two constraints.

Answer:
A very simple grammar that has these properties is $S \rightarrow Sa \mid b$. This grammar is SLR(1), but cannot be recognized by a recursive descent parser because it is left recursive.

- Prove that your grammar is SLR(1).
  Answer: The NFA of viable prefixes for this grammar is given by:



  Here, there is a potential shift/reduce conflict in State 1 on input a, however this conflict can be resolved by SLR(1) rules. Since a is not in Follow(start), the conflict is resolved in favor of a shift.

- Is your grammar also LR(0)? Why or why not?
  Answer: This grammar is not LR(0) because there is a shift/reduce conflict in state 1 between $S \rightarrow S.a$ (shift) and start $\rightarrow$ S. (reduction).

4. **Lexical Analysis and Parsing** (20 points)

After completing many tedious CS143 problem sets involving context free grammars, you have decided that you want to automate common tasks on grammars. To do this, you first need to write a lexer and parser for parsing valid context free grammars. You decide that the input grammars must obey the following specification:

- Every non-terminal starts with an uppercase, followed by any alphanumeric characters.
- Every terminal starts with a lowercase, followed by any alphanumeric characters.
- A valid grammar may contain any number (including 0) productions, separated by newlines.
- The right-hand-side of productions cannot be empty.

For example, the following is a valid input grammar:

```
Nt1 -> term1 Nt2 term2 | Nt3 | ..
Nt2 ->  term3 | ...
..
```

- Write a flex specification that recognizes the following tokens:

  `term`, the token representing terminals
  `non_term`, representing non-terminals
  `arrow`, representing $->$
  `sep`, representing |
  `end`, representing \n (newline)

  You do not need to provide actions to record the lexemes; simply indicate after your flex rule in {} which token is matched.

Answer:

```
[a-z][a-z,A-Z,0-9]*    { term }
[A-Z][a-z,A-Z,0-9]*    { non_term }
->                     { arrow }
|                      { sep }
\n                     { end }
```

- Assuming you have tokenized the input using your flex specification, give a grammar that recognizes CFG grammars.

  Answer:

  $$\begin{array}{lll} G & -> & \epsilon \mid P\ G \\ P & -> & non\_term\ arrow\ R\ end \\ R & -> & R\ sep\ U \ \mid U \\ U & -> & non\_term\ U \mid term\ U \mid non\_term \mid term \end{array}$$

- Is it possible for an input to have no lexer or parser error and still be malformed, in the sense that it is not a valid grammar? Justify your answer.

  Answer:

  Yes, consider the grammar:

  ```
  S -> A
  ```

  Here, A is undefined since no production contains A on its LHS. This is not a context free property and can therefore never be detected by the lexer or parser.