# CS302-2023-24: Mid-Semester Examination

Duration : 2 Hours — Max Marks : 70

- No explanations will be provided. In case of a doubt, make suitable assumptions and justify.
- There are no partial marks other than those defined in the evaluation rubrics.
- Please write only the final answers in your answer sheet. Rough work may be done at the back. Extra material (or multiple answers of the same questions) will receive negative marks.
- Answers must be in ink. Answers written using pencil will not be evaluated.

---

(1) Consider the following lex script. Give the output for the three inputs. **4+4+4=12**

```
%%
a            { printf("Token 1 (Lexeme %s)\n",yytext); }
a(b|cd)*b    { printf("Token 2 (Lexeme %s)\n",yytext); }
a(b|cd)*cd   { printf("Token 3 (Lexeme %s)\n",yytext); }
.            { ; }
```

(a) abbadbacdabbcdbb

(b) abbcdddabbbacdcdca

(c) abdddabbbacdcca

---

Answer:

The outputs are as shown below.

(a)
```
Token 2 (Lexeme abb)
Token 1 (Lexeme a)
Token 3 (Lexeme acd)
Token 2 (Lexeme abbcdbb)
```

(b)
```
Token 3 (Lexeme abbcd)
Token 2 (Lexeme abbb)
Token 3 (Lexeme acdcd)
Token 1 (Lexeme a)
```

(c)
```
Token 2 (Lexeme ab)
Token 2 (Lexeme abbb)
Token 3 (Lexeme acd)
Token 1 (Lexeme a)
```

Evaluation rubrics: Each completely correct line gets 1 mark.

---

(2) Consider the following grammar which generates **if** statements. For brevity, other statements are represented by $a$ and expressions derive $b$, with both being treated as terminals. Similarly, token **if** is represented by $i$, token **then** by $t$, and token **else** by $e$. **10+8=18**

$$\boxed{\begin{array}{l} S \to i\,E\,t\,S\,e\,S \\ S \to i\,E\,t\,S \\ S \to a \\ E \to b \end{array}}$$

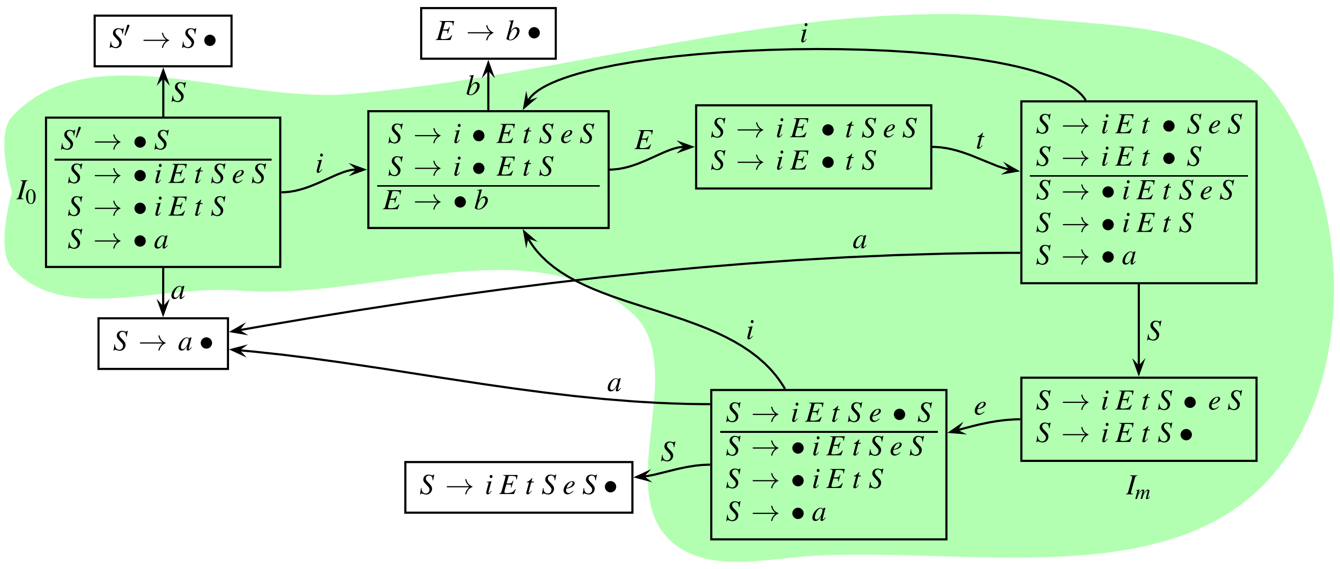
(a) Construct a DFA of LR(0) sets of items to explain why this grammar is not LR(0). Show all items in a state and separate closure items from kernel items by a horizontal line.

(b) Is the grammar ambiguous? If yes, draw two parse trees for the shortest sentence that demonstrates ambiguity. If no, explain why it is not ambiguous.

---

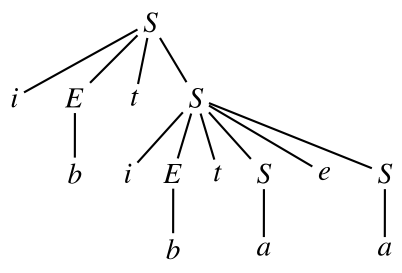
Answer:

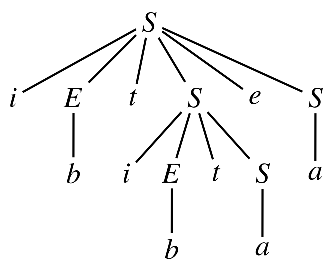
(a) The DFA of LR(0) items is shown below. State $i_m$ contains a complete item $S \to i\,E\,t\,S\,\bullet$ and an item $S \to i\,E\,t\,S\,\bullet\,e\,S$ that suggests a shift action on $e$. Since $e$ appears after $S$ in the RHS of $S \to i\,E\,t\,S\,e\,S$, $e \in FOLLOW(S)$ and hence the presence of the item $S \to i\,E\,t\,S\,\bullet$ suggests a reduction. This leads to a shift reduce conflict on $e$ in state $I_m$. Hence the grammar is not LR(0).



Evaluation rubrics: It is not necessary to show all the states. It is sufficient if all states appearing on all paths from $I_0$ to $I_m$ (along with the transitions that define these paths) are shown. Full marks if the shaded region below is correct and 0 marks if everything is correct but shift reduce conflict is invisible. If shift reduce conflict is visible, 1 mark will be deducted for every mistake.

(b) The grammar is ambiguous because we get the following two parse trees for the sentence $i\,b\,t\,i\,b\,t\,a\,e\,a$. This sentence can be seen to have two structures. The first parse tree below illustrates the structure when **else** is associated with the most closely nested **then** : $\boxed{i\,b\,t\,\boxed{i\,b\,t\,a\,e\,a}}$. The alternative structure, illustrated by the second parse tree, associates **else** with an outer **then** : $\boxed{i\,b\,t\,\boxed{i\,b\,t\,a}\,e\,a}$.





Evaluation rubrics: 4 marks for each tree. If there is any error in a tree, 0 marks for the tree.

---

(3) The following grammar models the syntax of the grammar rules in a yacc script. Non-terminals $L$, $P$, and $R$, represent a list of productions, a production, and an RHS of a production. For simplicity, we assume that a production $A : \alpha \mid \beta$ is written as a sequence of two productions $A : \alpha$ and $A : \beta$ and thus we do not use the metacharacter "|" for alternative RHS. Besides, without any loss of generality, we assume that every token is given a name and hence $R$ is just a sequence of *id*s. Note that yacc does not need a semicolon (";") to terminate a production. Hence it does not appear in our grammar. **10+8=18**

$$\boxed{\begin{array}{l} L \rightarrow L\,P \\ L \rightarrow P \\ P \rightarrow id : R \\ R \rightarrow R\ id \\ R \rightarrow id \end{array}}$$

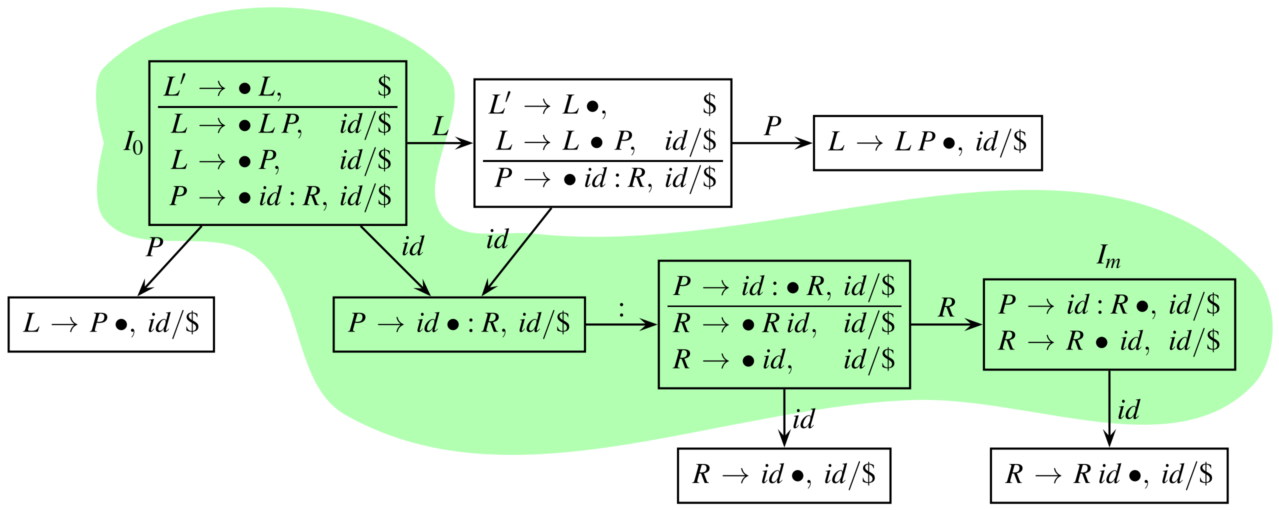
(a) Construct a DFA of LR(1) sets of items to explain why this grammar is not LR(1). Show all items in a state and separate closure items from kernel items by a horizontal line.

(b) Is the grammar ambiguous? If yes, draw two parse trees for the shortest sentence that demonstrates ambiguity. If no, explain why it is not ambiguous.

---

Answer:

(a) The DFA of LR(1) sets of items is as shown below. State $I_m$ contains a complete item $P \rightarrow id : R \bullet,\ id/\$$ suggesting a reduction on *id* but it also contains the item $R \rightarrow R \bullet id,\ id/\$$ which suggests a shift on *id*. Thus there is a shift reduce conflict in state $I_m$ and hence the grammar is not LR(1).



$I_0$
$L' \rightarrow \bullet L,\ \$$
$L \rightarrow \bullet L\,P,\ id/\$$
$L \rightarrow \bullet P,\ id/\$$
$P \rightarrow \bullet id : R,\ id/\$$
$L$
$L' \rightarrow L \bullet,\ \$$
$L \rightarrow L \bullet P,\ id/\$$
$P \rightarrow \bullet id : R,\ id/\$$
$P$
$L \rightarrow L\,P \bullet,\ id/\$$
$P$
$L \rightarrow P \bullet,\ id/\$$
$id$
$id$
$P \rightarrow id \bullet : R,\ id/\$$
$:$
$P \rightarrow id : \bullet R,\ id/\$$
$R \rightarrow \bullet R\ id,\ id/\$$
$R \rightarrow \bullet id,\ id/\$$
$R$
$I_m$
$P \rightarrow id : R \bullet,\ id/\$$
$R \rightarrow R \bullet id,\ id/\$$
$id$
$R \rightarrow id \bullet,\ id/\$$
$id$
$R \rightarrow R\ id \bullet,\ id/\$$

Evaluation rubrics: It is not necessary to show all the states. It is sufficient if all states appearing on all paths from $I_0$ to $I_m$ (along with the transitions that define these paths) are shown. Full marks if the shaded region below is correct and 0 marks if everything is correct but shift reduce conflict is invisible. If shift reduce conflict is visible, 1 mark will be deducted for every mistake.

(b) The grammar is unambiguous. Consider the sentence $id_1 : id_2\ id_3\ w$ where $w$ is a string of terminals.

The shift reduce conflict in state $I_m$ basically suggests that when have the viable prefix $id_1 : R$ on the stack and the next input symbol is $id_3$, there are two possible actions and it is not clear which one is correct:

- we could shift $id_3$ assuming it is a part of the RHS of the rule for $id_1$, or
- we could reduce $id_1 : R$ to $P$ (and $P$ to $L$) assuming that $id_3$ begins a new production.

Note that this depends on the first terminal in $w$. If it is a colon (":"), then $id_3$ begins a new production. On the other hand, if it is $id_4$, then $id_3$ is a part of an RHS of $id_1$. Both inputs $id_1 : id_2\ id_3\ id_4$ and $id_1 : id_2\ id_3 : id_4$ have a unique rightmost (and leftmost) derivation and hence a unique parse tree.

Thus the grammar is not ambiguous but is LR(2) (and not LR(1)) because it needs a lookahead of two symbols: a lookahead of $id_3 :$ suggests a reduction in state $I_m$ whereas a lookahead of $id_3\ id_4$ suggests a shift. This problem would vanish if a semicolon was mandatory for terminating a production.

Observe that it is ironic that although yacc generates an LALR(1) parser, it cannot generate a parser to parse its own script!

Evaluation rubrics: 8 marks for explaining that the grammar is not ambiguous but needs a look ahead of 2 to see if the following symbol is ":". 0 otherwise.

Some students have come up with alternative arguments for showing that the grammar is not ambiguous. Every valid argument will get full marks regardles of whether it mentions the need of a look ahead of two or not. Here's a summary of the alterntives suggested by the students

- Most students have uses some sort of inductive argument on the sentences to prove the unambiguity of (the regex and, consequently) the grammar. Some have made similar arguments about parse trees.
- One student has given a new parser to unambiguously parse each production.
- Some students have just mentioned or implied "more lookahead is needed," and that includes 2, so that also seems right.
- One student has written a proof-by-contradiction at the first point of difference.

(4) Show the three address code generated during bottom-up parsing for the following code fragment. Use the syntax-directed definitions covered in the class. Note that the semantic rules associated with a production are evaluated when the reduction for the production is performed. Use the temporaries `ti`, $i \geq 0$ and labels `Li`, $i \geq 0$ generated in the order of parsing. **10**

Note: Do not invent your own syntax-directed translation scheme.

```
sum=0;
j=0;
while (j<M)
{    i=0;
     while (i<N)
     {   sum = sum + A[i];
         i = i+1;
     }
     j = j+1;
}
```

---

Answer:

The generated 3-address code is as follows. During bottom-up parsing, the actions for the inner loop are carried out before those for the outer loop and hence the temporaries and labels are first generated for the inner loop.

```
    sum = 0;
    j = 0;
L2: t0 = j<M
    t8 = !t0
    if t8 goto L3
    i = 0;
L0: t1 = i<N;
    t6 = !t1
    if t6 goto L1
    t2 = i*4
    t3 = A[t2]
    t4 = sum+t3
    sum = t4
    t5 = i+1
    i = t5
    goto L0
L1: t7 = j+1
    j = t7
    goto L2
L3:
```

Evaluation rubrics: Each fully correct multi-line block in the following picture gets 2 marks provided all blocks appear in the right order.

The numbering of the temporaries have been corrected to make it consistent with the order of reductions. It now matches the ordering generated by sclp. We may give half marks if the code is completely correct except for the numbering of temporaries.

```
    sum = 0;
    j = 0;
L2: t0 = j<M
    t8 = !t0
    if t8 goto L3
------------------
    i = 0;
L0: t1 = i<N;
    t6 = !t1
    if t6 goto L1
-----------------
    t2 = i*4
    t3 = A[t2]
    t4 = sum+t3
    sum = t4
-----------------
    t5 = i+1
    i = t5
    goto L0
-----------------
L1: t7 = j+1
    j = t7
    goto L2
L3:
```

(5) Convert the syntax-directed definition used in question (4) into a syntax-directed translation scheme to generate the code for a **for** loop containing a **break** or a **continue** statements generated by the grammar given below. Recall that a **break** statement ignores the subsequent statements in the loop body and causes the control to exit the loop whereas a **continue** statement ignores the subsequent statements in the loop body and resumes the next iteration of the loop.

$$\boxed{\begin{array}{l} S \rightarrow \textbf{for}\ (E;\ E;\ E)\ S \\ S \rightarrow \textbf{break}\ ; \\ S \rightarrow \textbf{continue}\ ; \end{array}}$$

Define attributes $S.code$, $S.exit$, and $S.loopback$. Assume that the attributes $E.place$, $E.code$, and $S.code$ for other statements are computed outside of this grammar and are available to you. Assume the usual $gen$ function that takes a variable number of arguments and returns the generated code, and the "||" operator which joins two pieces of code. Leave as many attribute evaluations to the end of an RHS, as possible. **12**

---

Answer:

Note: Many of you pointed out the error in the answer given earlier in which the **continue** statement missed the increment code ($E_3.code$) and argued that three labels are needed. This is correct with the usual strategy of generation that we have been adopting in the class: We need a label for condition test, a label for loop exit, and a label for loop increment; the last lable is needed only to support the **continue** statement. The corrected SDTS with three labels is given below.

$$\boxed{\begin{array}{lll}
S_1 & \rightarrow & \textbf{for}\ (E_1;\ E_2;\ E_3) \\
 & & \{\ S_2.increment = getNewLabel()\ \ \text{/* needed here because it is inherited */} \\
 & & \quad S_2.loopback = getNewLabel()\ \ \text{/* can be moved to the end of the rule */} \\
 & & \quad S_2.exit = getNewLabel()\qquad \text{/* needed here because it is inherited */} \\
 & & \} \\
 & & S_2 \\
 & & \{\ t_1 = getNewTemp() \\
 & & \quad c_1 = gen(S_2.loopback, :) \\
 & & \quad c_2 = gen(t_1, =!, E_2.place)\ ||\ gen(\text{if}, t_1, \text{goto}, S_2.exit) \\
 & & \quad c_3 = gen(S_2.increment, :) \\
 & & \quad c_4 = gen(\text{goto}, S_2.loopback) \\
 & & \quad c_5 = gen(S_2.exit, :) \\
 & & \quad S_1.code = E_1.code\ ||\ c_1\ ||\ E_2.code\ ||\ c_2\ ||\ S_2.code\ ||\ c_3\ ||\ E_3.code\ ||\ c_4\ ||\ c_5 \\
 & & \} \\
S & \rightarrow & \textbf{break}\ ; \{S.code = gen(\text{goto}, S.exit)\} \\
S & \rightarrow & \textbf{continue}\ ; \{S.code = gen(\text{goto}, S.increment)\}
\end{array}}$$

Some students have come up with a clever strategy of generating the code that uses only two labels. They have achieved this by unrolling the loop once and putting the increment and condition test before the second occurrence of the loop body. This obviates the need of a separate label for condition test. The structure of the generated code looks as shown below:

```
      initialization
      condition test
      if the condition fails, goto L2
      loop body
L1:   increment
      condition test
      if the condition fails, goto L2
      loop body
      goto L1
L2:
```

In either case, inherited attributes are needed.

Some other students have used a technique that uses only synthesized attributes and compensates for the lack of inherited information by using *back-patching*. In a nut shell, it generates incomplete goto statements but records which statements require their labels to be filled in.

Revised evaluation rubrics: I view this, I have decided to use the following stragegy for giving marks:

- All students who have written a correct answer with two labels only (whether by unrolling or by back-patching) will get 4 marks as bonus (i.e., 16 marks) for this question.
- All students whose answers point out the need for three labels, will get 12 marks regarldless of the SDTS they have written.
- Other students who have attempted the question will get 8 marks regardles of what they have written.
- The students who have not attempted the question will get 4 marks.

Note that barely writing half a sentence or just the number of question does not amount to attempting the question. This decision will be taken subjectively and no discussion on this decision will be entertained.

---

Best Luck!