Midterm I Solutions
CS164, Spring 2006

February 23, 2006

- Please read all instructions (including these) carefully.
- **Write your name, login, SID, and circle the section time.**
- There are 8 pages in this exam and 4 questions, each with multiple parts. Some questions span multiple pages. All questions have some easy parts and some hard parts. If you get stuck on a question move on and come back to it later.
- You have 1 hour and 20 minutes to work on the exam.
- The exam is closed book, but you may refer to your two pages of handwritten 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. We might deduct points if your solution is far more complicated than necessary. Partial solutions will be graded for partial credit.

LOGIN: _____

NAME: _____

SID: _____

Circle the time of your section: Fri 10:00 Fri 11:00 Fri 2:00

Problem	Max points	Points
1	6	
2	30	
3	20	
4	44	
TOTAL	100	

1 Miscellaneous (6 points)

(a) Fill the missing right-hand side for the production of A in the grammar below such that the rules form a LL(1) grammar.

$$S \rightarrow A\ a\ A \mid A\ b\ A$$

$$A \rightarrow$$

Solution:

$$A \rightarrow \epsilon$$

Consider the following grammar:

$$S \rightarrow a\ S \mid b\ S \mid a\ b\ a\ A$$

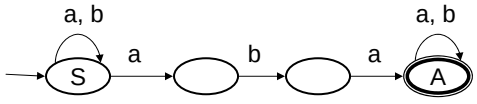
$$A \rightarrow \varepsilon \mid a\ A \mid b\ A$$

(b) Is the above grammar LL(1)? Why or why not?

Solution: It is not LL(1) because on non-terminal S and lookahead a there are two productions possible ($a\ S$ and $a\ b\ a\ A$).

(c) Show an NFA that accepts the language of this grammar.

Solution: The solution is shown below. You can solve this problem without figuring out the language before: each non-terminal corresponds to an NFA state, ϵ transitions mark the accepting states, each production corresponds to a transition. This works for all grammars whose productions have at most one non-terminal on the right-hand-side, occurring at the end.



2 Finite Automata and Regular Expressions (30 points)

(a) Write a regular expression for the language of strings over the alphabet $\Sigma = \{a, b\}$ that have an even number of occurrences of a (this includes 0 occurrences).

Solution: There are many correct solutions. Perhaps the simplest is:

$$(b \mid ab^*a)^*$$

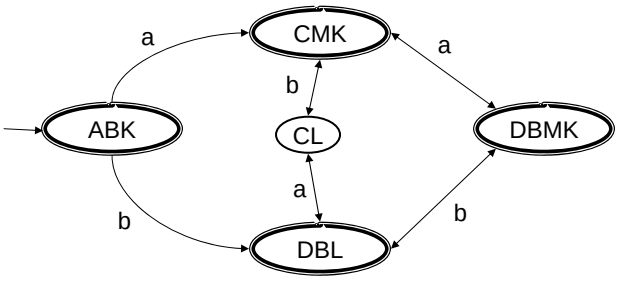
A close miss is

$$(ab^*a)^*$$

This one does not allow the string b.

(b) **[Long]** Draw the DFA corresponding to the following NFA. For each state in the resulting DFA write to which NFA states it corresponds.

Solution: The solution is shown below. Bi-directional edges stand for two edges going in opposite directions, with the same label.



In the following questions the alphabet is $\{0, 1\}$. Write TRUE if the language is regular and FALSE otherwise. Explain your answers in one or two sentences.

(c) The language of strings with no occurrences of the substring "010".

Solution: TRUE. The complement of a regular language is regular. Take the DFA that recognizes the 010 substring and swap rejecting and accepting states.

(d) The language of strings with more occurrences of 0 than 1.

Solution: FALSE. This would require counting up for a 0 and down for a 1. Unbounded counting cannot be done with finite automata.

(e) The language of strings whose length is a multiple of 3.

Solution: TRUE. Counting modulo N requires only N states.

(f) **[Harder]** The language of strings whose number of occurrences of 0 minus number of occurrences of 1 is a multiple of 3.

Solution: TRUE. This is also a counting modulo 3 problem, with counting up for a 0 and down for a 1.

(g) **[Harder]** The language of strings of terminals and non-terminals that can appear on the parsing stack during an LR(1) parse (stack is read from the bottom of the stack). Here the alphabet is the set of terminals and non-terminals of the grammar being parsed.

Solution: TRUE. After all we use a DFA to process the contents of the stack and decide whether to shift or to reduce. Take than DFA and make every state an accepting state.

3 LL Parsing (20 points)

Consider the following grammar with terminals `(`, `[`, `)`, and `]`.

$$\begin{array}{lcl} S & \rightarrow & TS \mid \texttt{[}S\texttt{]}S \mid \texttt{)}S \mid \epsilon \\ T & \rightarrow & \texttt{(}X\texttt{)} \\ X & \rightarrow & TX \mid \texttt{[}X\texttt{]}X \mid \epsilon \end{array}$$

(a) **[Long]** Fill in the table with the First and Follow sets for the non-terminals.

	First	Follow
S	([ ) ϵ	\$]
T	(	([ )] \$
X	([ϵ	)]

(b) **[Depends on (a)]** Fill in the LL(1) parsing table.

	`(`	`[`	`)`	`]`	\$
S	$T\ S$	[S] S	) S	ϵ	ϵ
T	(X)				
X	$T\ X$	[X] X	ϵ	ϵ	

(c) **[Depends on (b)]** Is this grammar LL(1)?

Solution: Yes, because every cell in the table above has at most one entry.

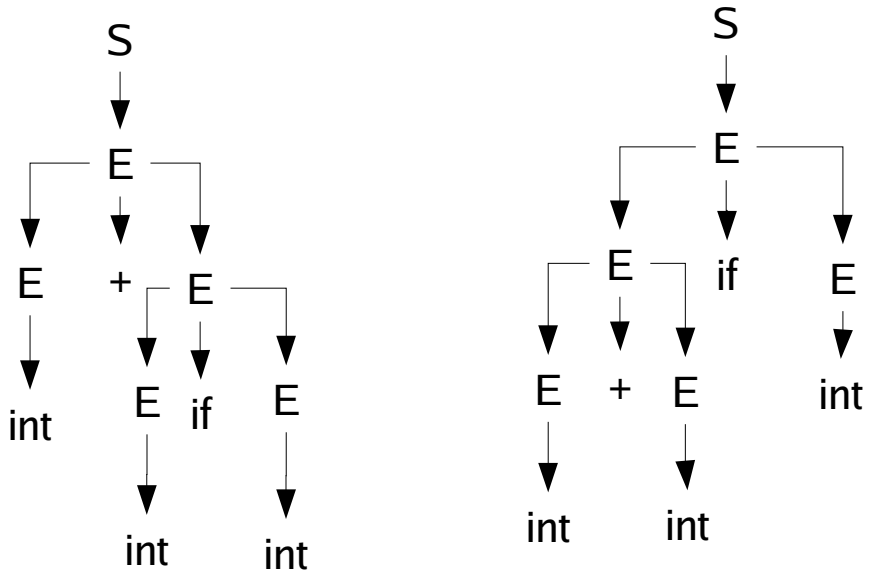
4 LR Parsing (44 points)

Consider the following grammar over terminals +, `if`, and `int`:

$$\begin{aligned} S &\rightarrow E \\ E &\rightarrow \text{int} \mid E \text{ if } E \mid E + E \end{aligned}$$

(a) This grammar is ambiguous. Show two parse trees for the string "`int` + `int` `if` `int`".

Solution:



On the next page there is a partially completed DFA for this grammar (state 0 is the initial state):

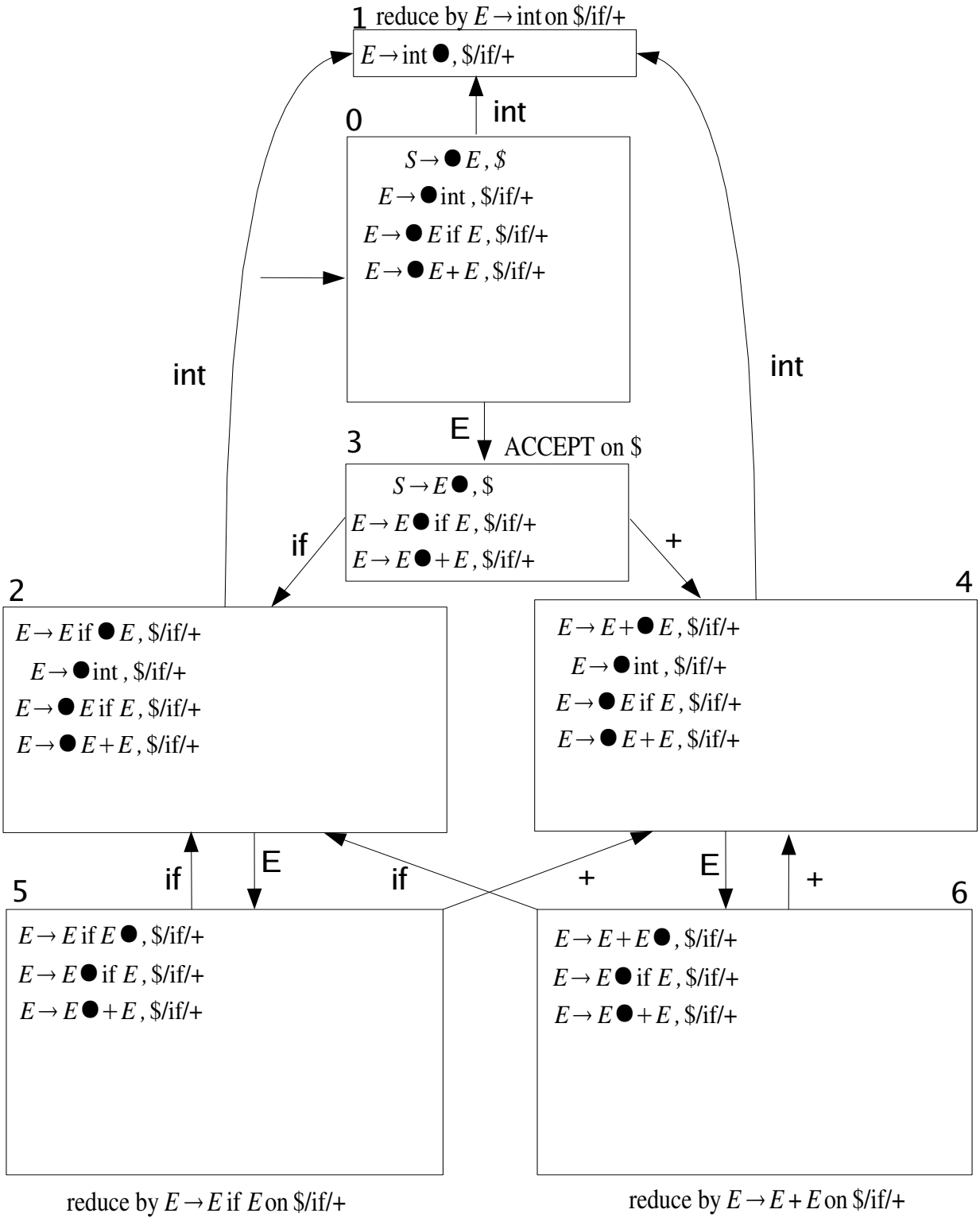
(b) **[Long]** Complete the DFA above. You need to do the following:

- Complete state 0 by performing closure on the item listed.
- Fill in all elements of states 2, 4, 5, 6, and the lookahead items for state 3.
- Fill in the missing transition labels on edges.
- Write the necessary "reduce-by" and "accept" labels on states.

(c) **[Depends on (b)]** For each state with a conflict, list the state, the lookahead token, and the type of conflict (i.e., shift-reduce conflict, or reduce-reduce conflict).

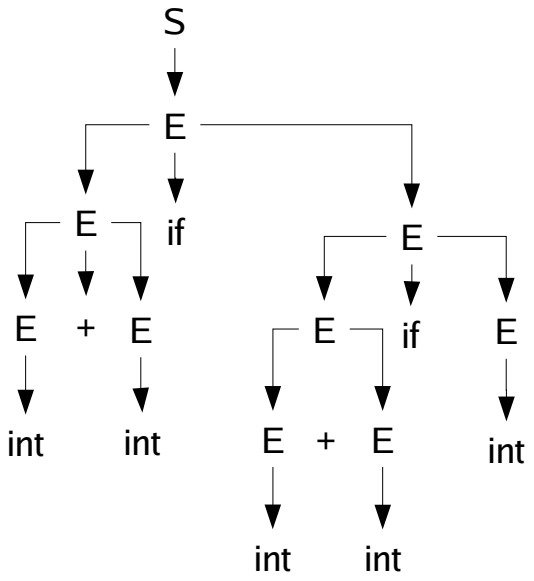
Solution: State 5, two conflicts: shift-reduce conflict on "if", shift-reduce conflict on "+".

State 6, two conflicts: shift-reduce conflict on "if", shift-reduce conflict on "+".


1 reduce by $E \rightarrow \text{int}$ on $/if/+
$E \rightarrow \text{int} \bullet$, $/if/+
int
0
$S \rightarrow \bullet E$, $
$E \rightarrow \bullet \text{int}$, $/if/+
$E \rightarrow \bullet E \text{ if } E$, $/if/+
$E \rightarrow \bullet E + E$, $/if/+
int
int
E
3
ACCEPT on $
$S \rightarrow E \bullet$, $
$E \rightarrow E \bullet \text{if } E$, $/if/+
$E \rightarrow E \bullet + E$, $/if/+
if
+
2
$E \rightarrow E \text{ if } \bullet E$, $/if/+
$E \rightarrow \bullet \text{int}$, $/if/+
$E \rightarrow \bullet E \text{ if } E$, $/if/+
$E \rightarrow \bullet E + E$, $/if/+
4
$E \rightarrow E + \bullet E$, $/if/+
$E \rightarrow \bullet \text{int}$, $/if/+
$E \rightarrow \bullet E \text{ if } E$, $/if/+
$E \rightarrow \bullet E + E$, $/if/+
if
E
if
+
E
+
5
$E \rightarrow E \text{ if } E \bullet$, $/if/+
$E \rightarrow E \bullet \text{if } E$, $/if/+
$E \rightarrow E \bullet + E$, $/if/+
6
$E \rightarrow E + E \bullet$, $/if/+
$E \rightarrow E \bullet \text{if } E$, $/if/+
$E \rightarrow E \bullet + E$, $/if/+
reduce by $E \rightarrow E \text{ if } E$ on $/if/+
reduce by $E \rightarrow E + E$ on $/if/+

(d) [**Does not depend on (b)**] Assume that we disambiguate the grammar by saying that + is left-associative and binds more tightly than `if`, and `if` is right associative. Show the correct parse tree for the string `int` + `int` `if` `int` + `int` `if` `int`

Solution:



(e) [**Depends on (b)**] For each of the conflicts that you have identified above, show how it must be resolved to achieve the disambiguation described above. For a shift/reduce conflict say whether we should shift or reduce. For a reduce/reduce conflict say which reduction we should choose.

Solution:

State 5: shift on "if", shift on "+".

State 6: reduce on "if", reduce on "+".