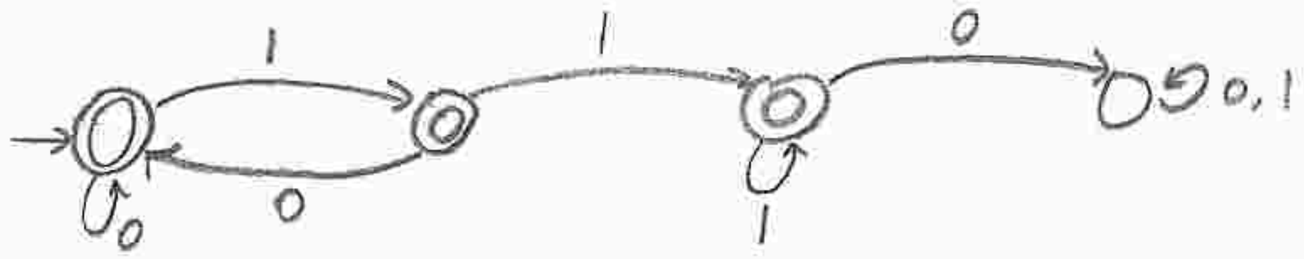
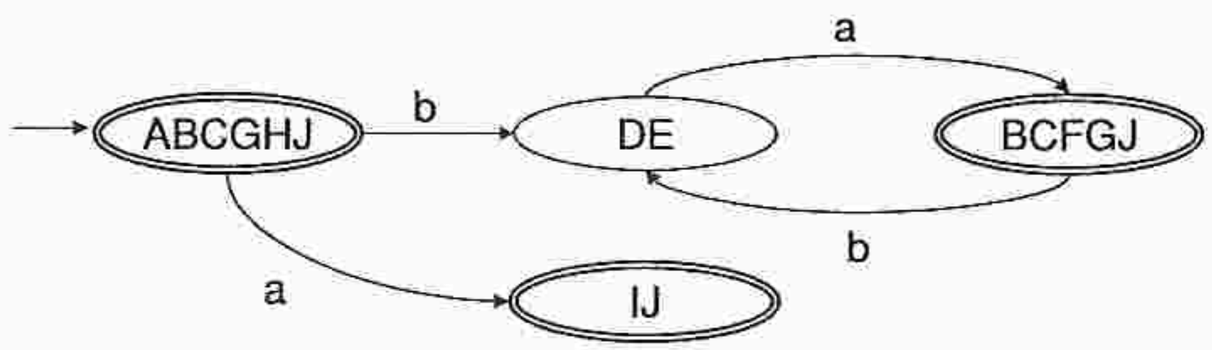
1 Finite Automata and Regular Expressions (20 points)

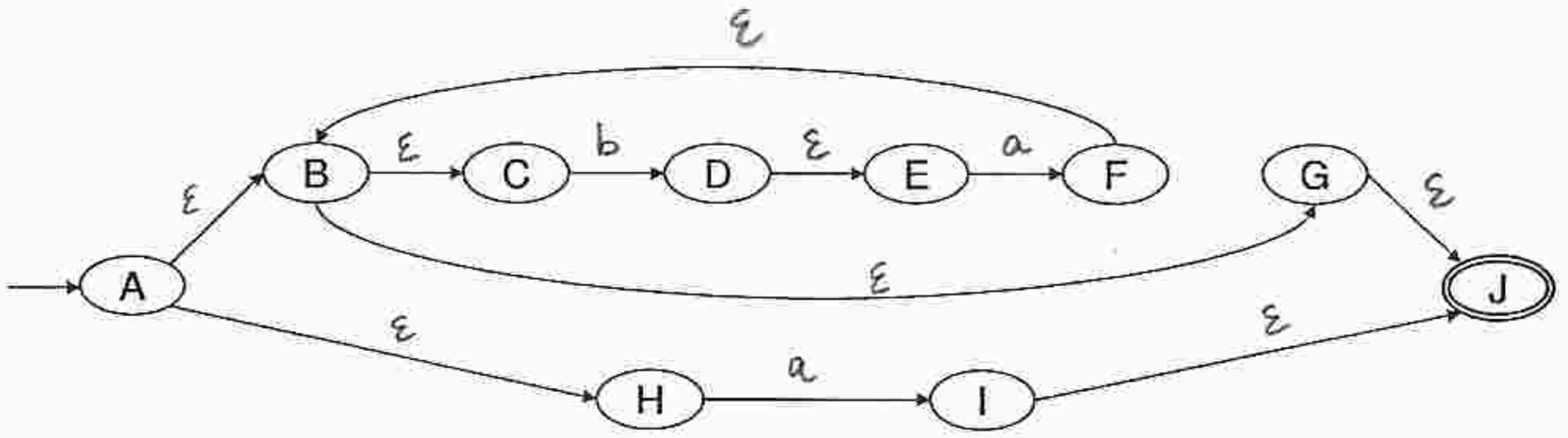
Spring 02 c) Draw a deterministic finite automaton(DFA) for the language of all strings over the alphabet{0,1} that do **not** contain the substring 110.



(b) Consider the following DFA over the alphabet $\Sigma = \{a, b\}$.



i. Label the transitions of the following NFA so that it accepts the same language as the DFA. Your NFA should transform to the given DFA by applying the NFA-to-DFA conversion algorithm given in lecture.

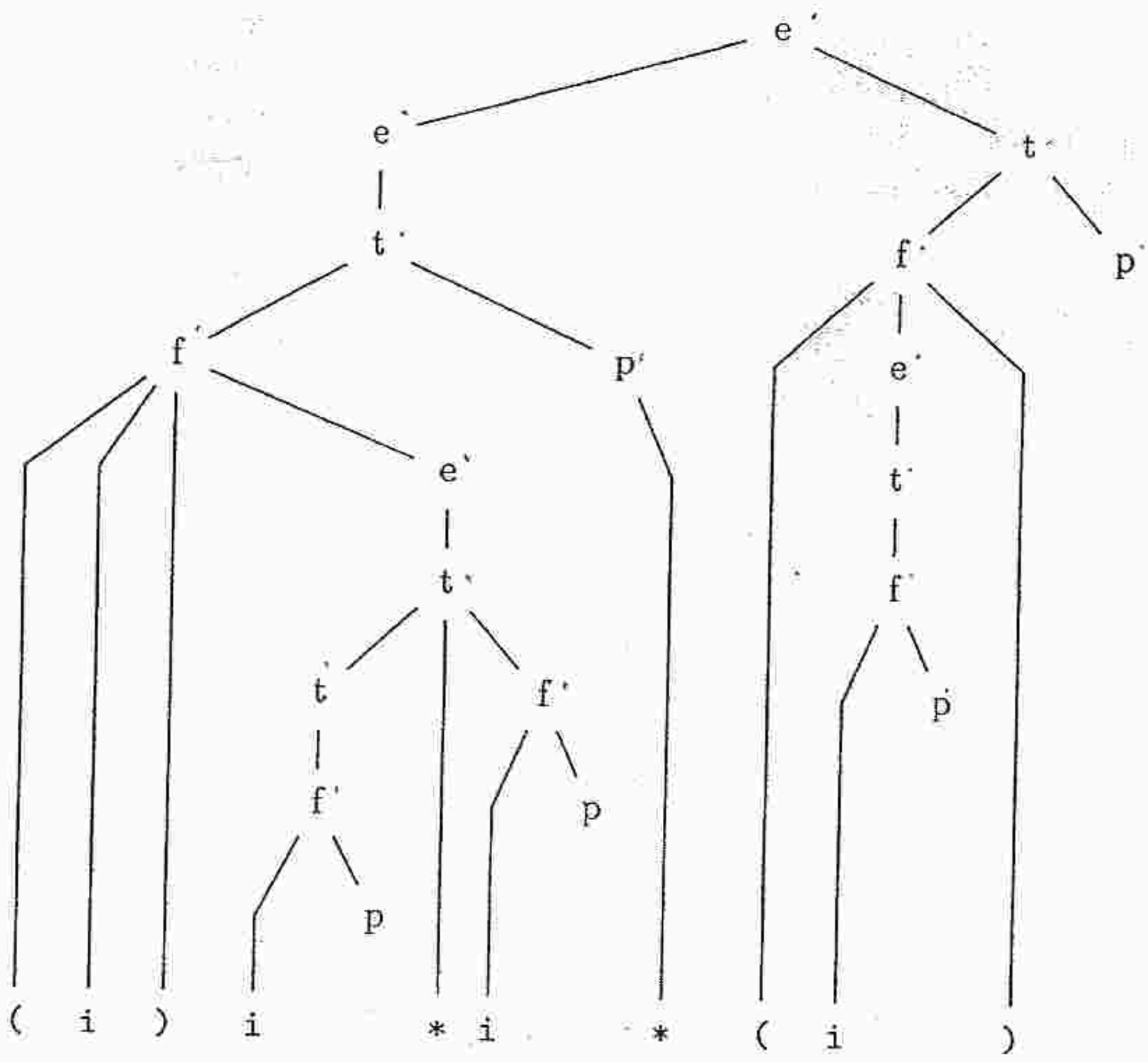


ii. Write the regular expression for this language. Among the several possible answers, write the one that would transform to this NFA by applying the regular expression-to-NFA conversion algorithm given in lecture.

$(ba)^* \mid a$

Spring 08

4. [8 points] Consider the parse tree below. All terminal symbols are written at the same level on the bottom row.



a. Show as much of the grammar as one can deduce from this parse tree.

$e \rightarrow e\ t \mid t$

$t \rightarrow f\ p \mid t * f \mid f$

$f \rightarrow (\ i\ )\ e \mid i\ p \mid (\ e\ )$

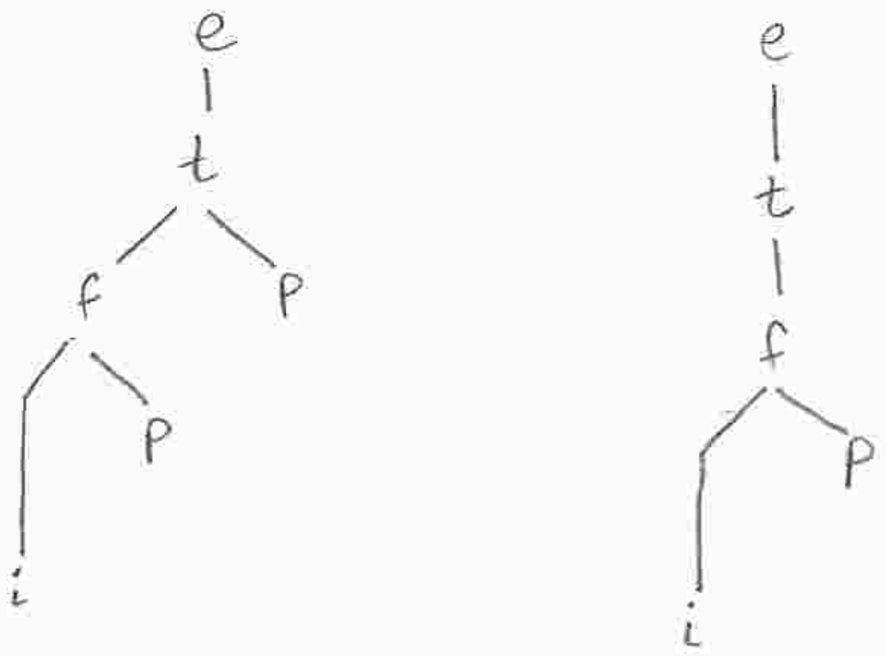
$p \rightarrow \varepsilon \mid *$

More questions on the next page

b. Show a reverse rightmost derivation corresponding to this tree, as performed by a shift-reduce parser. (However, you need not mention the shifts and reductions, just the sentential forms that get you back to the start symbol from the input).

(i) i _ * i * (i)
⇒ (i) i p * i * (i)
⇒ (i) f * i * (i)
⇒ (i) t * i _ * (i)
⇒ (i) t * i p * (i)
⇒ (i) t * f * (i)
⇒ (i) t * (i)
⇒ (i) e * (i)
⇒ f * (i)
⇒ f p (i)
⇒ t (i)
⇒ e (i _)
⇒ e (i p)
⇒ e (f)
⇒ e (t)
⇒ e (e)
⇒ e f _
⇒ e f p
⇒ e t
⇒ e

c. Show that the grammar is ambiguous by giving a string that has (at least) two parses and giving two distinct parse trees for it.



Continued on next page.

d. Given one of the parse trees for an ambiguous string in this language, can you find two leftmost derivations that give rise to that parse tree? If so, show such a tree and show the derivations; otherwise say why you cannot.

No. Each parse tree corresponds to exactly one leftmost derivation.

Spring 05

Login:

1. [8 points] Suppose that string literals consist of one or more "stringlets" separated from each other by whitespace (one or more characters, each one of which is either a blank or a newline). Each stringlet consists of any sequence of characters (including newlines) between quotes ("), but with any occurrence of a quotation mark inside the string doubled. For example, "" and """" are valid stringlets – the first denoting the empty string and the second denoting one quotation mark. Thus the following are valid string literals:

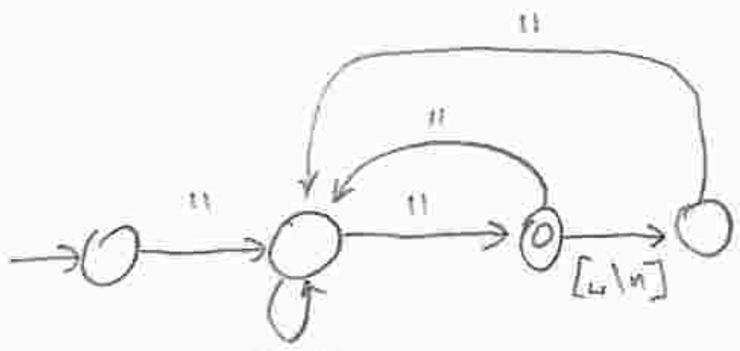
1. "Hello, " "world."
2. "I said, ""Hi"""
3. "These are
   two lines"
4. "These are
   "
   "two" " " "lines"

a. Give a regular expression for these literals, using Lex notation (including definitions, if you want).

STRINGLET \" ([^"]*|\"\")*\"

{STRINGLET} ([␣\n]+{STRINGLET})*

b. Produce the simplest DFA (yes, it must be deterministic) you can for these literals.



Fall 04

2. First/Follow/LL(1)

This question concerns the context-free grammar given below (where capital letters denote non-terminals, small letters denote terminals).

$$E \rightarrow Z\,X\,b \mid Z\,a$$
$$X \rightarrow d\,Z \mid \in$$
$$Z \rightarrow a\,X\,X \mid X$$

Part a).

Fill in the following table with First and Follow sets for the grammar.

α	*FIRST* (α)	*FOLLOW* (α)
E	a, b, d	$
X	d, ε	a, b, d
Z	a, d, ε	a, b, d
$Z\,X\,b$	a, b, d	
$Z\,a$	a, d	
$d\,Z$	d	
$a\,X\,X$	a	
X	d, ε	
$\in$	ε	

Part b).

In the *LL*(1) parsing table below, fill in the row corresponding to the non-terminal A:

	a	*b*	*d*	$
Z	Z → aXX Z → X	Z → X	Z → X	

Part c).

Is the grammar *LL*(1)? Justify your answer.

No. Multiple Productions are in entry for a.

Part d).

Suppose that in the above grammar, you replace $\in$ with c, so the grammar becomes

$$E \rightarrow Z\,X\,b \mid Z\,a$$
$$X \rightarrow d\,Z \mid c$$
$$Z \rightarrow a\,X\,X \mid X$$

Now, you are parsing the string **a c c d c b** with a non-deterministic LR parser. For each of the following stack/input configurations, state whether or not they can lead to a successful parse (YES/NO will suffice).

a c c | d c b No

a X X | d c b Yes

a X X d Z | b No

a X X X | b No

Z X | b Yes

Only one parse tree: