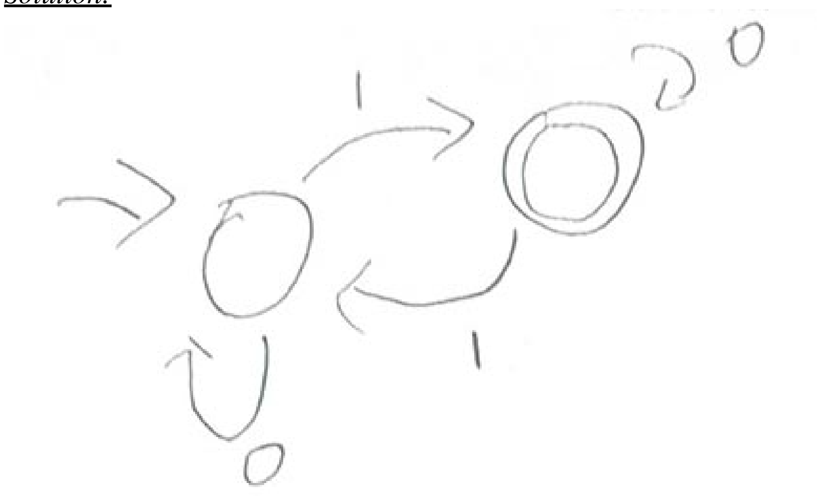
CS164 - Midterm 1 Spring 2002

1 Regular Expressions and Finite Automata (20 pts)

a) Draw a deterministic finite automaton (DFA) that recognizes the language over the alphabet {0,1} consisting of all those strings that contain an odd number of 1's.

Solution:



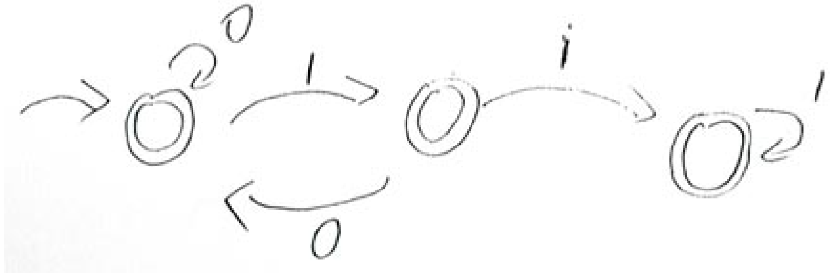
b) Write a regular expression for this language.

Solution:

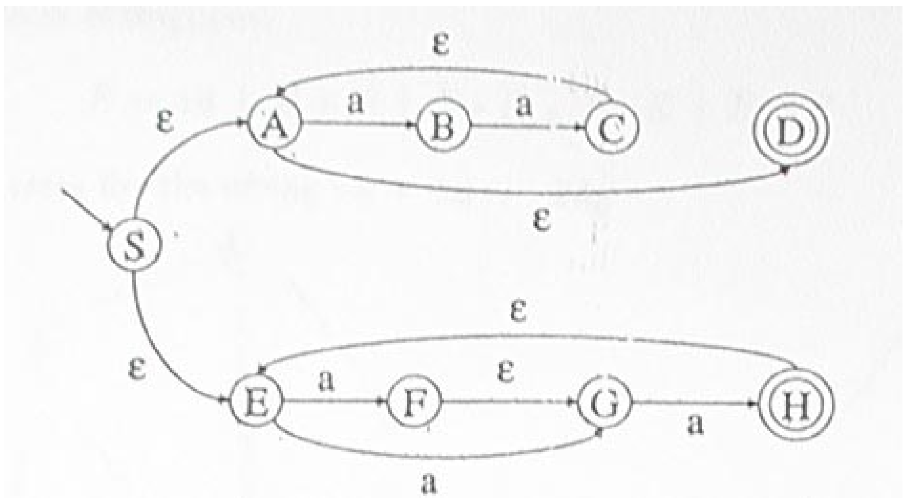
0*1(0*10*1)*0*

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.

Solution:

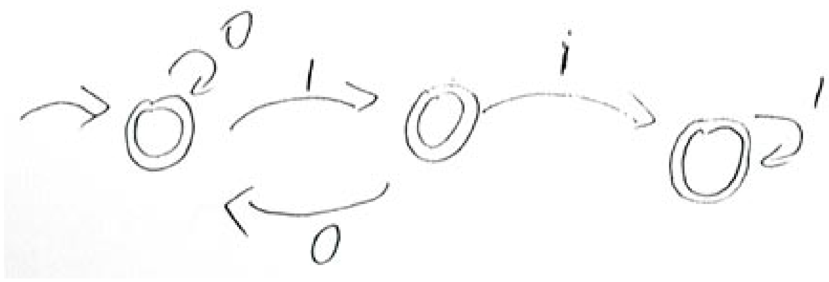


d) Consider the following NFA over the alphabet a}. Convert this NFA to a DFA. For each DFA state write the set of the NFA states that it corresponds to.



Solution:

	State	a	(notes)
1	SAED	BGF	1 = states SAED = ADES
2	BFG	CAHED	2 = states BFG = BFG
3	LAHED	BFG	3 = states CAHED = ACDEH



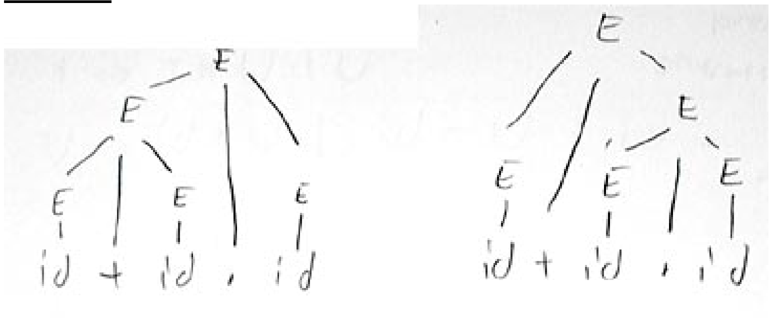
2 Context-Free Grammars (20 pts)

The following grammar is ambiguous:

E --> id | E + E | E * E | E . E | E - E

a) Show two parse trees for the string id + id . id

Solution:

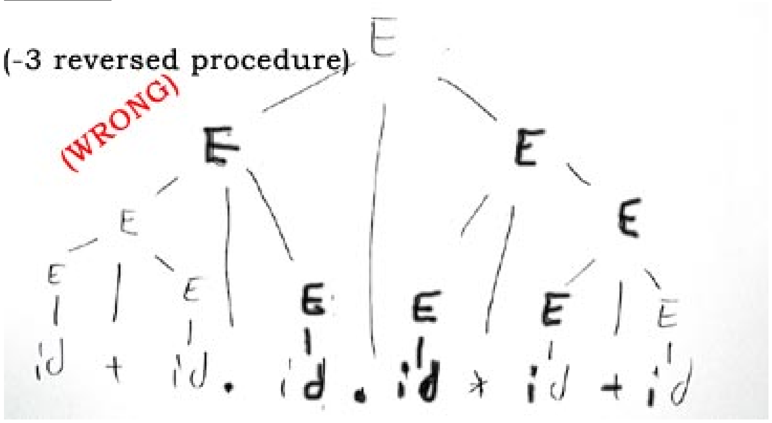


b) The ambiguity is removed by adding the following precedence declarations. Recall the convention that the first declaration assigns the lowest precedence:

%right + -
%left *
%left .

Considering these declarations, show the parse tree for the string:
id + id . id . id * id + id

Solution:

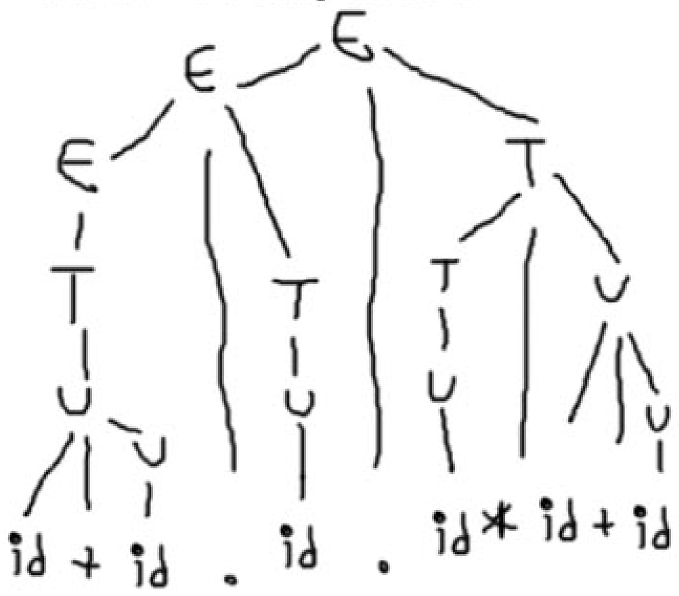


c) Write an unambiguous grammar for the same language, resolving ambiguity as specified by the precedence declarations shown above.

Solution:

E -> E . T | T
T -> T * U | U
U -> id + U | id - U | id

Note. Still reversed procedure.



3 LL Parsing (15 pts)

a) Consider the following grammar over the alphabet {a, b, d}:

S -> A B D
A -> a | B S B
B -> b | D
D -> d | E

Fill in the table below with the First and Follow sets for the non-terminals in this grammar:

Solution:

	First	**Follow**
S	a, b, d	$, b, d
A	a, b, d	b, d, $
B	b, d, E	a, b, d, $
D	d, E	a, b, d, $

F:(s) = F:(A)
F:(A) = a F:(B) F:(S)
F:(B) = b F:(D)
F:(D) = d | E

Fo(S) = Fi(B), Fo(A)
Fo(A) = Fi(B), Fi(D), Fo(S)
Fo(B) = Fi(S), Fi(D), Fo(A)

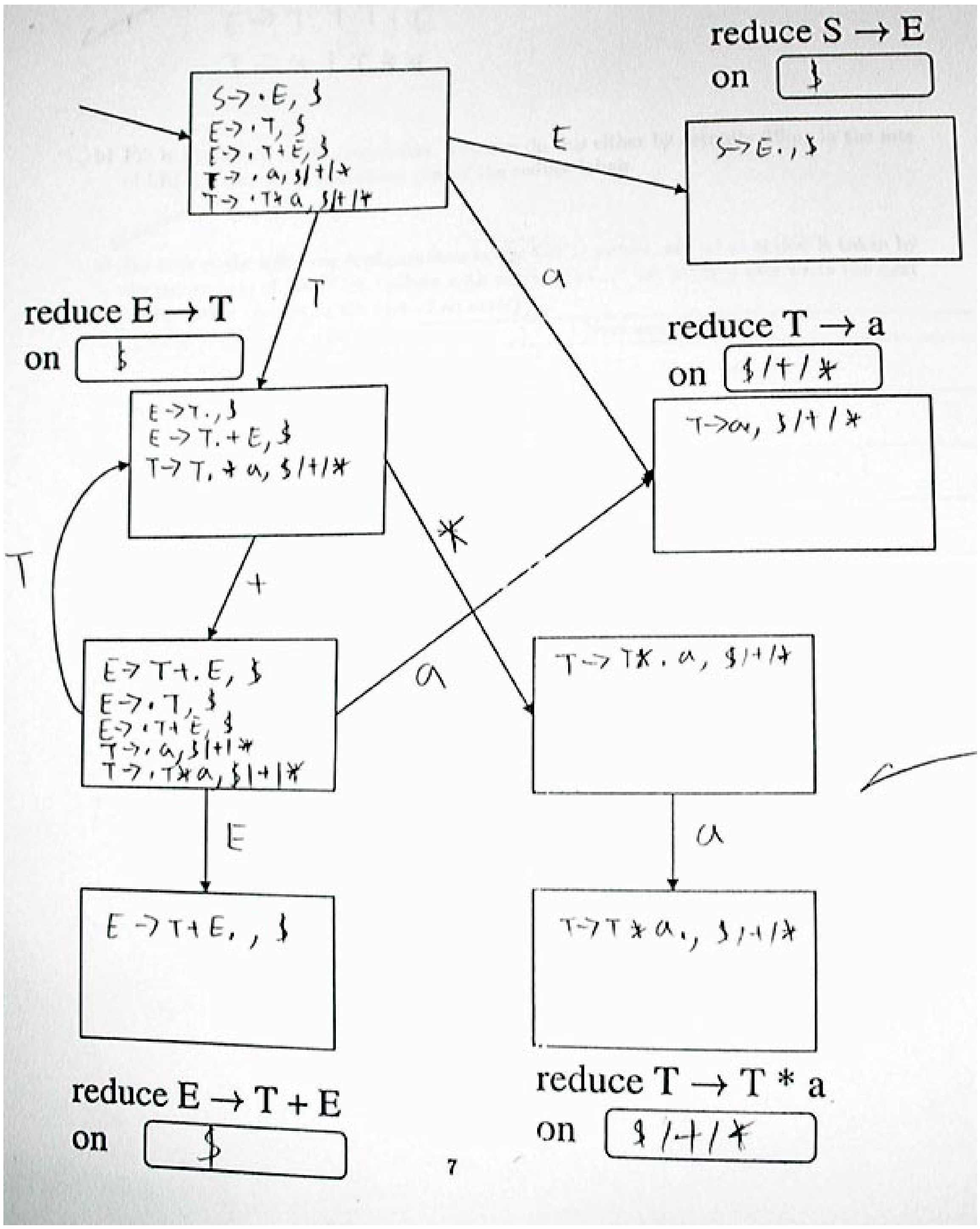
Fo(D) = Fo(B), Fo(S)

---

4 LR Parsing (30 pts)

The following figure shows the LR(1) DFA for a grammar. The DFA is missing several elements that you'll have to fill-in: the LR(1) items that describe each state, the labels on the transitions and the lookahead terminals for the reduce labels.

Solution:



a) Write the grammar that corresponds to the above DFA.

Solution:

S -> E
E -> T | T + E
T -> a | T * a

4 LR Parsing (30 pts)

b) Fill in the labels on the transitions. You can do this either by actually filling in the sets of LR(1) items or by examining closely the reduce labels.

Solution:

Done

c) For each of the following configurations of the LR(1) parser, say what action is taken by the parser (one of "shift" or "reduce with production ..." or "error") and write the next configuration (except in the case of an error).

Solution:

Configuration	Action	Next-configuration
T > +a + a$	shift	T + > a + a $
T + > * a$	error	
T + T + a > *a$	reduce T -> a	T + T + T > * a $
T * a > $	reduce T -> T * a	T > $
T + E > + a$	error	

d) Fill in the large boxes (representing states) with the LR(1) items

Solution:

Done

e) Fill in the lookahead terminals in the small boxes next to each reduce label.

Solution:

Done

5 LL Parsing (15 pts)

Consider the following grammar in which S is the start non-terminal and in which the productions for the non-terminals A, B, and C are not shown. There are no other productions for S.

S -> A B | A C
A -> ...
B -> ...
C -> ...

Consider now the LL(1) parsing table for this grammar. What conditions must be satisfied by the First and Follow sets
the non-terminals, such that the row corresponding to *S* in this table **does not** contain multiply-defined entries?

Write your answer in the table below considering six separate cases. For example, in the top-left entry consider the case when E *is not in the set* First(b) and E *is not in the set* First(C) and First(A) = {E}, and write what **additional** conditions must be met, or write "never" if no additional conditions would help.

Solution:

	E *not in* First(B) E *not in* First(C)	E *not in* First(B) E *in* First(C)	E *in* First(B) E *in* First(C)
First(A) = {E}	First(B) and First(C) does not have common terminals.	First(B) + First(C) cannot have common terminals and Follow(S) cannot have common terminal as First(b) or First(c)	First(B) + First(C) cannot have common terminals. Follow(S) cannot have common terminals w/First(B) or First(C) as well.
First(A) != {E}	Never	Never	Never