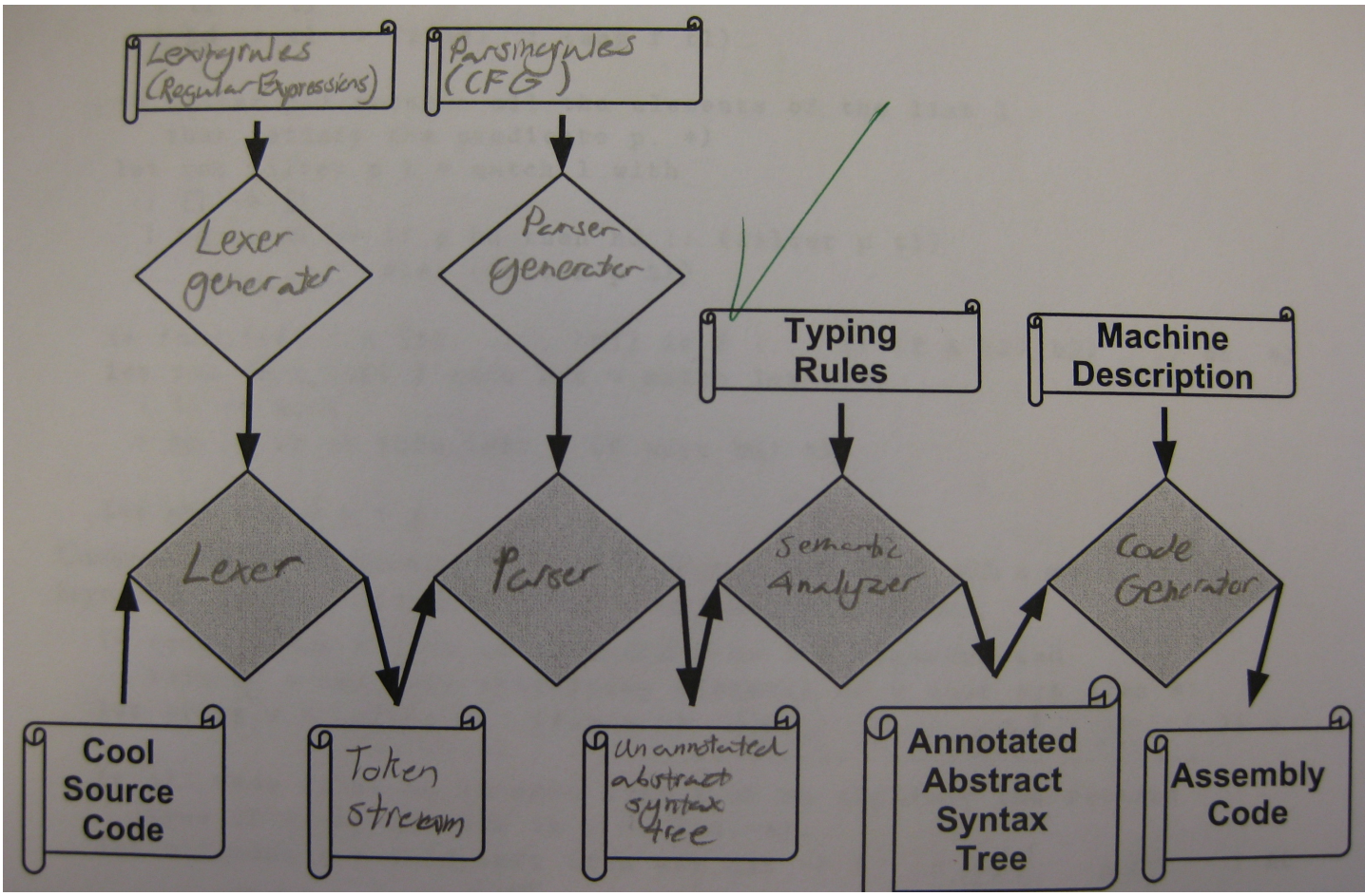



```
(* evens takes a list of integers w as its argument and
   returns a new list containing elements of w that are even *)
let evens w = filter (fun x -> not (is_odd x))

(* all_odds takes an integer list x as an argument and returns
   true if every element in x is odd. *)
let all_odds x = fold_left (fun acc elt -> ( is_odd elt )
    acc ) true x

(* prod_list takes an integer list y as an argument and returns
   the arithmetic product of all of the elements of y *)
let prod_list y = fold_left mul 1 y

(* inner_prod [ [1;2;3] ; [] ; [7;8] ] returns [6; 1; 56].  *)
let inner_prod z = map prod_list z
```

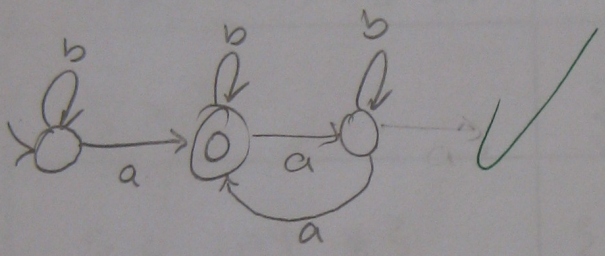
3 Regular Expressions and Automata (19 points)

For this question, the regular expressions are single character (a), epsilon (ϵ), concatenation ($r_1 r_2$), disjunction ($r_1|r_2$), Kleene star $r*$, plus $r+$ and option $r?$.

(a) (6 pts.) Write a regular expression (over the alphabet $\Sigma = \{a, b\}$) for the language of strings that have an odd (and non-zero) number of occurrences of a (but may contain other characters).

$b^* a (b^* a b^* a b^*)^*$ ✓

(b) (6 pts.) Draw a **DFA** that accepts the language from the above problem.



✓

(c) (1 pt.) Always, Sometimes or Never. Given a regular expression r, there exists a DFA d such that $L(r) = L(d)$. equiv

(d) (1 pt.) Always, Sometimes or Never. Given a regular expression r, there exists a NFA n such that $L(r) = L(n)$. add $\bigcirc \xrightarrow{\epsilon} \bigcirc$

(e) (1 pt.) Always, Sometimes or Never. Given a regular expression r, there exists a PDA p such that $L(r) = L(p)$. don't use stack

(f) (1 pt.) Always, Sometimes or Never. Given a context-free grammar g, there exists a DFA d such that $L(g) = L(d)$. if CFG doesn't use stack

(g) (1 pt.) Always, Sometimes or Never. Given a finite language L_1, there exists a regular expression r such that $L_1 = L(r)$. enum all and $(x|y|z)$ them

(h) (1 pt.) Always, Sometimes or Never. Given a regular expression r that contains $*$ but neither ϵ nor ?, there exists a finite language L_2 such that $L_2 = L(r)$. → infinite

(i) (1 pt.) Always, Sometimes or Never. Given an NFA n, there exists a regular expression r containing neither $+$ nor ? such that $L(n) = L(r)$. DFAs and NFAs are equivalent in expressive power.

Consider the following grammar with four terminals: =, +, *, int.

$$
\begin{aligned}
S &\rightarrow B + B \\
A &\rightarrow * \\
B &\rightarrow \epsilon \\
B &\rightarrow \texttt{int}\ B\ B \\
B &\rightarrow A = \\
\end{aligned}
$$

(a) (10 pts.) Fill in the table with the First and Follow sets for the non-terminals.

	First	Follow
S	int, *, +	$
A	*	=
B	ε, int, *	int, *, +, $

(b) [Depends on 4(a).] (8 pts.) Fill in the LL(1) parsing table.

	=	+	*	int	$
S		B+B	B+B	B+B	
A			*		
B		ε	ε / A =	ε / int B B	ε

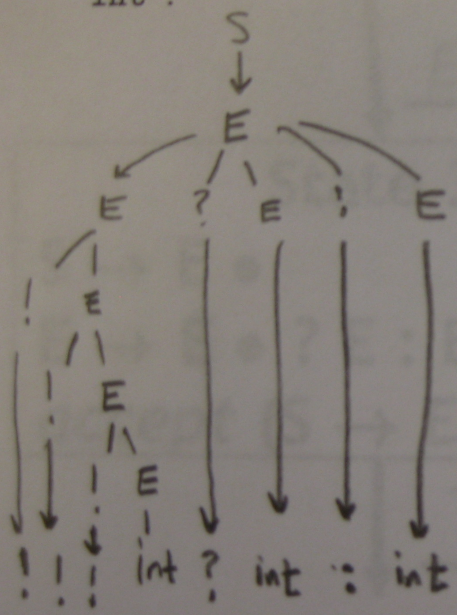
(c) [Depends on 4(b).] (2 pts.) Is the grammar LL(1)? Why or why not?

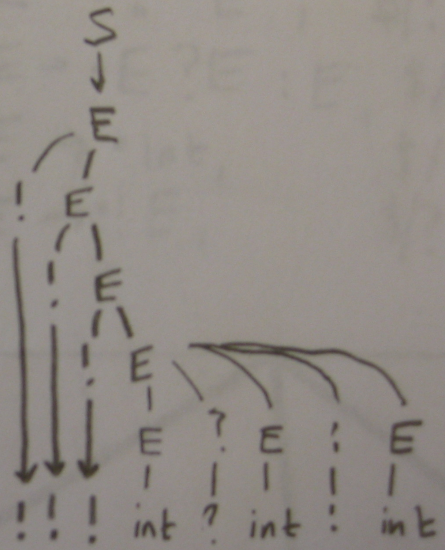
No. T[B][*] and T[B][int] are multiply defined.

NICE

(a) (10 pts.) Show that this grammar is ambiguous using the string "! ! ! int ? int".

!!! int ? int : int





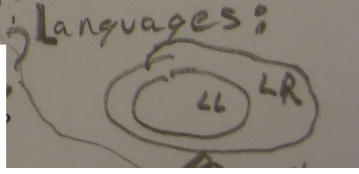
(c) [Depends on 5(b).] (4 pts.) For each state with a conflict: list the state, the lookahead token, and the type of conflict (i.e., shift-reduce or reduce-reduce).

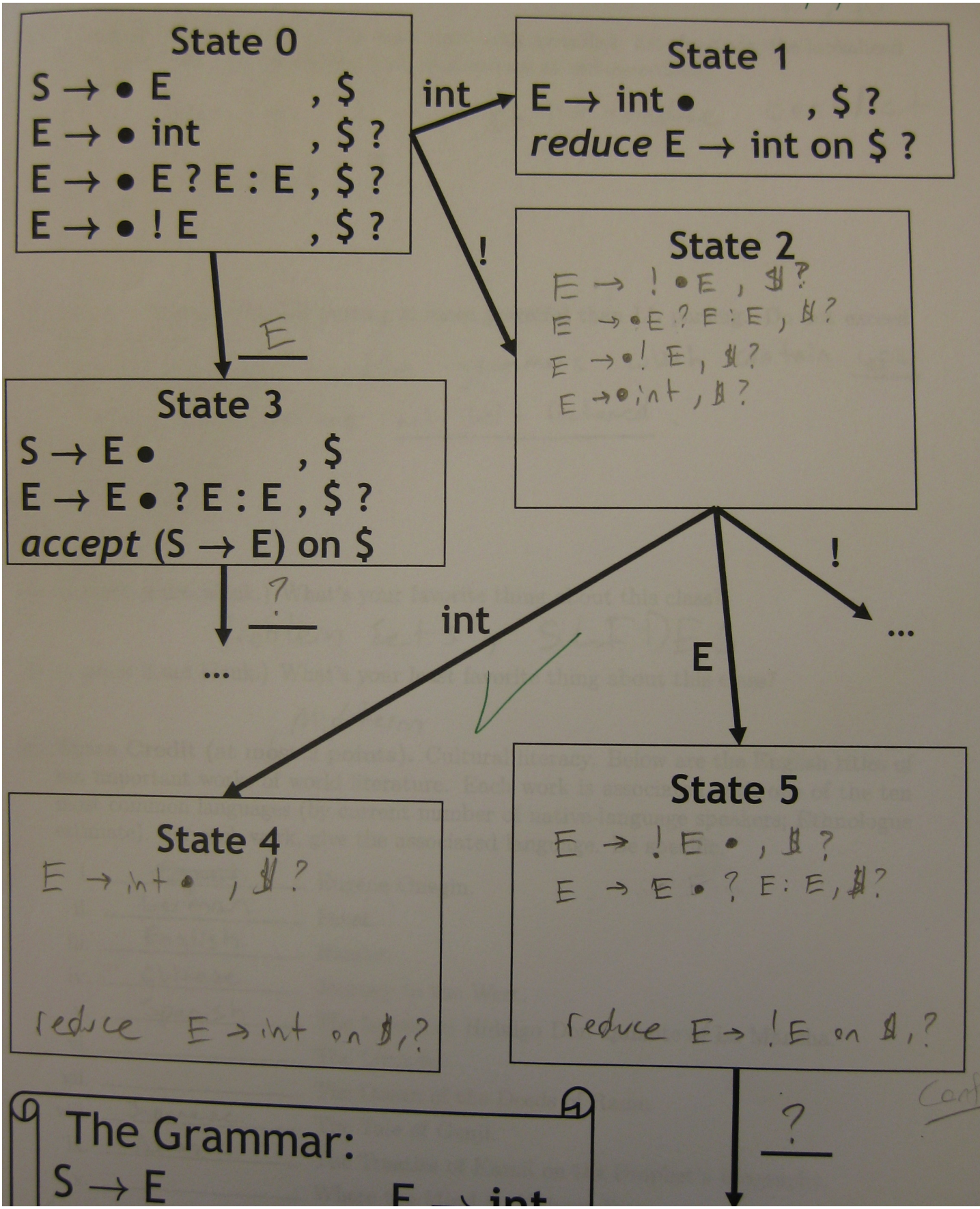
state 5 has a shift-reduce conflict with lookahead token ? ✓

(d) (4 pts.) Explain why LR parsing is more powerful than LL parsing. Do not exceed four sentences.

LL gramars must be left-factored and left-recursion most be eliminated. often this is difficult to do with programming langvages. The languages LR can parse are more numerous that LL, in fact LR can parse all LL languages but not neccessarily the opposite.

(e) (1 point if not blank.) What's your favorite thing about this class?




State 0
S → • E , $
E → • int , $?
E → • E ? E : E , $?
E → • ! E , $?
int
!
E
State 1
E → int • , $?
reduce E → int on $?
State 2
E → ! • E , $?
E → • E ? E : E , $?
E → • ! E , $?
E → • int , $?
State 3
S → E • , $
E → E • ? E : E , $?
accept (S → E) on $
?
...
int
E
!
...
State 4
E → int • , $?
reduce E → int on $?
State 5
E → ! E • , $?
E → E • ? E : E , $?
reduce E → ! E on $?
?
The Grammar:
S → E