(* 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)) w

(* 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 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 (r₁r₂), disjunction (r₁|r₂), Kleene star r*, plus r+ and option r?.

(a) (6 pts.) Write a regular expression (over the alphabet Σ = {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 \( L(r) = L(d) \).

(d) (1 pt.) Always, Sometimes or Never. Given a regular expression r, there exists an NFA n such that \( L(r) = L(n) \).

(e) (1 pt.) Always, Sometimes or Never. Given a regular expression r, there exists a PDA p such that \( L(r) = L(p) \).

(f) (1 pt.) Always, Sometimes or Never. Given a context-free grammar g, there exists a DFA d such that \( L(g) = L(d) \).

(g) (1 pt.) Always, Sometimes or Never. Given a finite language \( L_1 \), there exists a regular expression r such that \( L_1 = L(r) \) 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 \( L_2 = L(r) \).

(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: $=, +, *, \text{int}$.

$$
S \rightarrow B + B \\
A \rightarrow * \\
B \rightarrow \epsilon \\
B \rightarrow \text{int} B B \\
B \rightarrow A =
$$

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

<table>
<thead>
<tr>
<th></th>
<th>First</th>
<th>Follow</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>int, A, +</td>
<td>$$</td>
</tr>
<tr>
<td>A</td>
<td>$\epsilon$</td>
<td>$\epsilon$, int, A, +, $$</td>
</tr>
<tr>
<td>B</td>
<td>$\epsilon$, int, A</td>
<td>$\epsilon$, int, A, +, $$</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>$=$</th>
<th>$+$</th>
<th>$*$</th>
<th>int</th>
<th>$$</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>$B + B$</td>
<td>$B + B$</td>
<td>$B + B$</td>
<td>$S \rightarrow B + B$</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>$\epsilon$</td>
<td>$\epsilon$</td>
<td>$\epsilon$</td>
<td>$\epsilon$, int B B,</td>
<td>$\epsilon$</td>
</tr>
<tr>
<td>B</td>
<td>$\epsilon$</td>
<td>$\epsilon$</td>
<td>$\epsilon$</td>
<td>$\epsilon$, int B B,</td>
<td>$\epsilon$</td>
</tr>
</tbody>
</table>

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

*No*, $T[B][A]$ and $T[B][\text{int}]$ are multiply defined.
(a) (10 pts.) Show that this grammar is ambiguous using the string "!!! 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 grammars must be left-factored and left-recursion must be eliminated. Often this is difficult to do with programming languages. The languages LR can parse are more numerous than LL, in fact LR can parse all LL languages but not necessarily the opposite.

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