```
open List;;
open Pervasives;;

let rec subset a b =
    match a with
    | [] -> true
    | head::tail -> (
        match (mem head b) with
        | true -> (subset tail b)
        | false -> false
    )
;;

let equal_sets a b =
    (subset a b) && (subset b a)
;;

let rec set_union a b =
    match a with
    | [] -> b
    | head::tail-> (
        match (mem head b) with
        | true -> set_union tail b
        | false -> set_union tail (head::b))
;;

let rec set_intersection a b =
    match a with
    | [] -> []
    | head::tail -> (
        match (mem head b) with
        | true -> head::(set_intersection tail b)
        | false -> set_intersection tail b )
;;

let rec set_diff a b =
    match a with
    | []->[]
    | head::tail ->(
        match (mem head b) with
        | true -> set_diff tail b
        | false -> head::(set_diff tail b))
;;


let rec computed_fixed_point eq f x =
    match (eq (f x) x) with
    | true -> x
    | false -> computed_fixed_point eq f (f x)
;;


let rec is_periodic eq f p x orig =
    match p with
    | 0 -> true
    | 1 -> (eq (f x) orig)
    | _ -> is_periodic eq f (p-1) (f x) orig

(*define anonymous function that recursevly calls itself with p*)
(*have computed period point call itself with x = (f x)*)
let rec computed_periodic_point eq f p x =
    match (is_periodic eq f p x x) with
```

```
    | true -> x
    | false -> (computed_periodic_point eq f p (f x))
;;

let rec while_away s p x =
    match (p x) with
    | true -> x::(while_away s p (s x))
    | false -> []
;;

(* another implementation of computed_fixed_point*)
(*let computed_fixed_point eq f x = computed_periodic_point eq f 1 x;; *)
let rec rle_decode lp =
    let rec rep_symbol n s =
        match n with
        | 0 -> []
        | _ -> s::(rep_symbol (n-1) s)
    in
    match lp with
    | [] -> []
    | (num,sym)::tail -> append (rep_symbol num sym) (rle_decode tail)
;;


type ('nonterminal, 'terminal) symbol =
  | N of 'nonterminal
  | T of 'terminal


let rec find_sub_terminals rhs =
    match rhs with
    | [] -> []
    | (T some_term)::tail -> (T some_term)::(find_sub_terminals tail)
    | (N some_non_term)::tail2 -> (find_sub_terminals tail2)
(*create a list of all terminals present*)

let rec find_terminals grammars =
    match grammars with
    | [] -> []
    | (lhs, rhs)::tail1 -> (
        match rhs with
        | [] -> find_terminals tail1
        | (N non_term)::tail2 -> set_union (find_sub_terminals tail2) (find_terminals
tail1)
        | (T term)::tail3 -> set_union (find_sub_terminals tail3) (set_union [T term]
(find_terminals tail1))
        )

let rec find_good_nonterms depth terms grammars =
    (*scan list for rules in the set of good non terminals and terminals*)
    let rec scan_list terms grammars =
    match grammars with
    | [] -> terms
    | (lhs, rhs)::tail -> (
            match (subset rhs terms) with
            | true -> set_union [N lhs] (if mem (N lhs) terms then
                    (scan_list terms tail) else scan_list (set_union [N lhs] terms)
grammars)
            | false -> scan_list terms tail )
        in
        match depth with
        | 0 -> terms
```

```
        | _ -> find_good_nonterms (depth -1) (scan_list terms grammars) grammars

let rec remove_blind_alley good_stuff grammars =
    match grammars with
        | [] -> []
        | (lhs, rhs)::tail -> (
            match (subset rhs good_stuff) with
            | true -> (lhs, rhs)::(remove_blind_alley good_stuff tail)
            | false -> remove_blind_alley good_stuff tail )

let filter_blind_alleys g =
    match g with
        | (start, rules) ->
        (start, remove_blind_alley (find_good_nonterms (length rules) (find_terminals
rules) rules) rules)
        (*create a tuple of start and good grammars,
        good grammars defined by finding all the terminals,
        finding all the non terminals connected to the terminals,
        then removing all rules not in the set composed of terminals and good
non_terminals*)
;;

let subset_test0 = subset [] [1;2;3]
let subset_test1 = subset [3;1;3] [1;2;3]
let subset_test2 = not (subset [1;3;7] [4;1;3])

let equal_sets_test0 = equal_sets [1;3] [3;1;3]
let equal_sets_test1 = not (equal_sets [1;3;4] [3;1;3])

let set_union_test0 = equal_sets (set_union [] [1;2;3]) [1;2;3]
let set_union_test1 = equal_sets (set_union [3;1;3] [1;2;3]) [1;2;3]
let set_union_test2 = equal_sets (set_union [] []) []

let set_intersection_test0 =
  equal_sets (set_intersection [] [1;2;3]) []
let set_intersection_test1 =
  equal_sets (set_intersection [3;1;3] [1;2;3]) [1;3]
let set_intersection_test2 =
  equal_sets (set_intersection [1;2;3;4] [3;1;2;4]) [4;3;2;1]

let set_diff_test0 = equal_sets (set_diff [1;3] [1;4;3;1]) []
let set_diff_test1 = equal_sets (set_diff [4;3;1;1;3] [1;3]) [4]
let set_diff_test2 = equal_sets (set_diff [4;3;1] []) [1;3;4]
let set_diff_test3 = equal_sets (set_diff [] [4;3;1]) []

let computed_fixed_point_test0 =
  computed_fixed_point (=) (fun x -> x / 2) 1000000000 = 0
let computed_fixed_point_test1 =
  computed_fixed_point (=) (fun x -> x *. 2.) 1. = infinity
let computed_fixed_point_test2 =
  computed_fixed_point (=) sqrt 10. = 1.
let computed_fixed_point_test3 =
  ((computed_fixed_point (fun x y -> abs_float (x -. y) < 1.)
             (fun x -> x /. 2.)
             10.)
   = 1.25)

let computed_periodic_point_test0 =
  computed_periodic_point (=) (fun x -> x / 2) 0 (-1) = -1
let computed_periodic_point_test1 =
  computed_periodic_point (=) (fun x -> x *. x -. 1.) 2 0.5 = -1.
```

```
(* An example grammar for a small subset of Awk, derived from but not
   identical to the grammar in
   <http://web.cs.ucla.edu/classes/winter06/cs132/hw/hw1.html>.  *)

type awksub_nonterminals =
  | Expr | Lvalue | Incrop | Binop | Num

let awksub_rules =
   [Expr, [T"("; N Expr; T")"];
    Expr, [N Num];
    Expr, [N Expr; N Binop; N Expr];
    Expr, [N Lvalue];
    Expr, [N Incrop; N Lvalue];
    Expr, [N Lvalue; N Incrop];
    Lvalue, [T"$"; N Expr];
    Incrop, [T"++"];
    Incrop, [T"--"];
    Binop, [T"+"];
    Binop, [T"-"];
    Num, [T"0"];
    Num, [T"1"];
    Num, [T"2"];
    Num, [T"3"];
    Num, [T"4"];
    Num, [T"5"];
    Num, [T"6"];
    Num, [T"7"];
    Num, [T"8"];
    Num, [T"9"]]

let awksub_grammar = Expr, awksub_rules

let awksub_test0 =
  filter_blind_alleys awksub_grammar = awksub_grammar

let awksub_test1 =
  filter_blind_alleys (Expr, List.tl awksub_rules) = (Expr, List.tl awksub_rules)

let awksub_test2 =
  filter_blind_alleys (Expr,
      [Expr, [N Num];
       Expr, [N Lvalue];
       Expr, [N Expr; N Lvalue];
       Expr, [N Lvalue; N Expr];
       Expr, [N Expr; N Binop; N Expr];
       Lvalue, [N Lvalue; N Expr];
       Lvalue, [N Expr; N Lvalue];
       Lvalue, [N Incrop; N Lvalue];
       Lvalue, [N Lvalue; N Incrop];
       Incrop, [T"++"]; Incrop, [T"--"];
       Binop, [T"+"]; Binop, [T"-"];
       Num, [T"0"]; Num, [T"1"]; Num, [T"2"]; Num, [T"3"]; Num, [T"4"];
       Num, [T"5"]; Num, [T"6"]; Num, [T"7"]; Num, [T"8"]; Num, [T"9"]])
  = (Expr,
     [Expr, [N Num];
      Expr, [N Expr; N Binop; N Expr];
      Incrop, [T"++"]; Incrop, [T"--"];
      Binop, [T "+"]; Binop, [T "-"];
      Num, [T "0"]; Num, [T "1"]; Num, [T "2"]; Num, [T "3"]; Num, [T "4"];
      Num, [T "5"]; Num, [T "6"]; Num, [T "7"]; Num, [T "8"]; Num, [T "9"]])

let awksub_test3 =
```

```
  filter_blind_alleys (Expr, List.tl (List.tl (List.tl awksub_rules))) =
    filter_blind_alleys (Expr, List.tl (List.tl awksub_rules))

type giant_nonterminals =
  | Conversation | Sentence | Grunt | Snore | Shout | Quiet

let giant_grammar =
  Conversation,
  [Snore, [T"ZZZ"];
   Quiet, [];
   Grunt, [T"khrgh"];
   Shout, [T"aooogah!"];
   Sentence, [N Quiet];
   Sentence, [N Grunt];
   Sentence, [N Shout];
   Conversation, [N Snore];
   Conversation, [N Sentence; T","; N Conversation]]

let giant_test0 =
  filter_blind_alleys giant_grammar = giant_grammar

let giant_test1 =
  filter_blind_alleys (Sentence, List.tl (snd giant_grammar)) =
    (Sentence,
     [Quiet, []; Grunt, [T "khrgh"]; Shout, [T "aooogah!"];
      Sentence, [N Quiet]; Sentence, [N Grunt]; Sentence, [N Shout]])

let giant_test2 =
  filter_blind_alleys (Sentence, List.tl (List.tl (snd giant_grammar))) =
    (Sentence,
     [Grunt, [T "khrgh"]; Shout, [T "aooogah!"];
      Sentence, [N Grunt]; Sentence, [N Shout]])


let my_subset_test0 = subset [1;2;3] [1;2;3]
let my_subset_test1 = subset [[1;2;3]] [[1;2;3];[1;2]]
let my_subset_test2 = not (subset ["one";"two";"three"] ["four";"five";"six"])
let my_subset_test3 = subset ["one";"two";"three"]
["one";"two";"three";"four";"five";"six"]

let my_equal_sets_test0 = equal_sets [] []
let my_equal_sets_test1 = not (equal_sets [] [1;2;3])
let my_equal_sets_test2 = equal_sets ["one";"two";"three"] ["one";"two";"three"]

let my_set_union_test0 = equal_sets (set_union ["words"] ["words";"werds";"wurdz"] )
["words";"werds";"wurdz"]
let my_set_union_test1 = equal_sets (set_union [1;2] []) [1;2]
let my_set_union_test2 = equal_sets (set_union [["subset"];["sub";"subset"]] [["hello
subsets"]]) [["subset"];["sub";"subset"];["hello subsets"]]

let my_set_intersection_test0 = equal_sets (set_intersection ["set1"] ["set2"]) []
let my_set_intersection_test1 = equal_sets (set_intersection [2;5;0;6;2;4]
[1;8;0;0;2;7;3;8;2;5;5]) [2;5;0]
let my_set_intersection_test2 = equal_sets (set_intersection ["25"; "or"; "6"; "to"; "4"]
["to";"be";"or";"not"]) ["or";"to"]

let my_set_diff_test0 = equal_sets (set_diff ["bold";"as";"love"] ["love";"me";"do"])
["bold"; "as"]
let my_set_diff_test1 = equal_sets (set_diff [2;5;0;6;2;4] [1;8;0;0;2;7;3;8;2;5;5]) [6;4]
let my_set_diff_test2 = equal_sets (set_diff [1;2;3] [3;2;1]) []

let my_computed_fixed_point_test0 = ((computed_fixed_point (=) (fun fn -> fn + 0) 1) = 1)
```

```
let my_computed_fixed_point_test1 = ((computed_fixed_point (=) (fun fn -> fn / fn) 700034)
= 1)
let my_computed_fixed_point_test2 = ((computed_fixed_point (=) (fun fn -> (fn *. fn)) 8.)
= infinity)

let my_computed_periodic_point_test0 = ((computed_periodic_point (=) (fun fn -> fn) 0 1) =
1)
let my_computed_periodic_point_test1 = ((computed_periodic_point (=) (fun fn -> (fn *.
fn)) 1 0.2) = 0.)

let my_while_away_test0 = (equal_sets (while_away (fun x -> x-2) (fun fn -> ((fn mod 2) =
0) && (fn > 0)) 10) [2;4;6;8;10])
let my_while_away_test0 = (equal_sets (while_away (fun x -> x-2) (fun fn -> ((fn mod 4) =
0) && (fn > 0)) 10) [])


let my_rle_decode_test0 = (equal_sets (rle_decode [2,2;4,4]) [2;2;4;4;4;4])
let my_rle_decode_test1 = (equal_sets (rle_decode [1,"and";2,"on";2,"and on";1,"we go"])
["and" ;"on" ;"on" ;"and on" ;"and on" ;"we go"])


type just_college_thoughts =
    | Freshmen | Sophomore | Happiness | Sad | Depression | Expletive | Dropout


let college_thoughts_grammar =
    Freshmen,
    [Freshmen, [N Happiness];
    Freshmen, [N Sad];
    Freshmen, [N Happiness; N Sad; T "confusion"];
    Freshmen, [N Sophomore];
    Sophomore, [N Sophomore];
    Sophomore, [N Sad; N Depression];
    Sophomore, [N Depression];
    Depression, [N Depression];
    Sad, [N Expletive];
    Expletive, [N Dropout; T "welp"];
    Expletive, [T "small bad word"];
    Expletive, [T "big bad word"];
    Expletive, [T "many bad word"];
    Dropout, [T "such is life"]]

let my_blind_alley_test0 = (filter_blind_alleys college_thoughts_grammar
    =
    (Freshmen,
    [Freshmen, [N Sad];
    Sad, [N Expletive];
    Expletive, [N Dropout; T "welp"];
    Expletive, [T "small bad word"];
    Expletive, [T "big bad word"];
    Expletive, [T "many bad word"];
    Dropout, [T "such is life"]]))





# #use "hw1 - Copy.ml";;
val subset : 'a list -> 'a list -> bool = <fun>
val equal_sets : 'a list -> 'a list -> bool = <fun>
val set_union : 'a list -> 'a list -> 'a list = <fun>
val set_intersection : 'a list -> 'a list -> 'a list = <fun>
```

```
val set_diff : 'a list -> 'a list -> 'a list = <fun>
val computed_fixed_point : ('a -> 'a -> bool) -> ('a -> 'a) -> 'a -> 'a =
  <fun>
val is_periodic : ('a -> 'b -> bool) -> ('a -> 'a) -> int -> 'a -> 'b -> bool =
  <fun>
val computed_periodic_point :
  ('a -> 'a -> bool) -> ('a -> 'a) -> int -> 'a -> 'a = <fun>
val while_away : ('a -> 'a) -> ('a -> bool) -> 'a -> 'a list = <fun>
val rle_decode : (int * 'a) list -> 'a list = <fun>
type ('nonterminal, 'terminal) symbol = N of 'nonterminal | T of 'terminal
val find_sub_terminals : ('a, 'b) symbol list -> ('c, 'b) symbol list = <fun>
val find_terminals : ('a * ('b, 'c) symbol list) list -> ('d, 'c) symbol list =
  <fun>
val find_good_nonterms :
  int ->
  ('a, 'b) symbol list ->
  ('a * ('a, 'b) symbol list) list -> ('a, 'b) symbol list = <fun>
val remove_blind_alley :
  'a list -> ('b * 'a list) list -> ('b * 'a list) list = <fun>
val filter_blind_alleys :
  'a * ('b * ('b, 'c) symbol list) list ->
  'a * ('b * ('b, 'c) symbol list) list = <fun>
# #use "hw1sample.ml";;
val subset_test0 : bool = true
val subset_test1 : bool = true
val subset_test2 : bool = true
val equal_sets_test0 : bool = true
val equal_sets_test1 : bool = true
val set_union_test0 : bool = true
val set_union_test1 : bool = true
val set_union_test2 : bool = true
val set_intersection_test0 : bool = true
val set_intersection_test1 : bool = true
val set_intersection_test2 : bool = true
val set_diff_test0 : bool = true
val set_diff_test1 : bool = true
val set_diff_test2 : bool = true
val set_diff_test3 : bool = true
val computed_fixed_point_test0 : bool = true
val computed_fixed_point_test1 : bool = true
val computed_fixed_point_test2 : bool = true
val computed_fixed_point_test3 : bool = true
val computed_periodic_point_test0 : bool = true
val computed_periodic_point_test1 : bool = true
type awksub_nonterminals = Expr | Lvalue | Incrop | Binop | Num
val awksub_rules :
  (awksub_nonterminals * (awksub_nonterminals, string) symbol list) list =
  [(Expr, [T "("; N Expr; T ")"]); (Expr, [N Num]);
   (Expr, [N Expr; N Binop; N Expr]); (Expr, [N Lvalue]);
   (Expr, [N Incrop; N Lvalue]); (Expr, [N Lvalue; N Incrop]);
   (Lvalue, [T "$"; N Expr]); (Incrop, [T "++"]); (Incrop, [T "--"]);
   (Binop, [T "+"]); (Binop, [T "-"]); (Num, [T "0"]); (Num, [T "1"]);
   (Num, [T "2"]); (Num, [T "3"]); (Num, [T "4"]); (Num, [T "5"]);
   (Num, [T "6"]); (Num, [T "7"]); (Num, [T "8"]); (Num, [T "9"])]
val awksub_grammar :
  awksub_nonterminals *
  (awksub_nonterminals * (awksub_nonterminals, string) symbol list) list =
  (Expr,
   [(Expr, [T "("; N Expr; T ")"]); (Expr, [N Num]);
    (Expr, [N Expr; N Binop; N Expr]); (Expr, [N Lvalue]);
    (Expr, [N Incrop; N Lvalue]); (Expr, [N Lvalue; N Incrop]);
    (Lvalue, [T "$"; N Expr]); (Incrop, [T "++"]); (Incrop, [T "--"]);
```

```
    (Binop, [T "+"]); (Binop, [T "-"]); (Num, [T "0"]); (Num, [T "1"]);
    (Num, [T "2"]); (Num, [T "3"]); (Num, [T "4"]); (Num, [T "5"]);
    (Num, [T "6"]); (Num, [T "7"]); (Num, [T "8"]); (Num, [T "9"])])
val awksub_test0 : bool = true
val awksub_test1 : bool = true
val awksub_test2 : bool = true
val awksub_test3 : bool = true
type giant_nonterminals =
    Conversation
  | Sentence
  | Grunt
  | Snore
  | Shout
  | Quiet
val giant_grammar :
  giant_nonterminals *
  (giant_nonterminals * (giant_nonterminals, string) symbol list) list =
  (Conversation,
   [(Snore, [T "ZZZ"]); (Quiet, []); (Grunt, [T "khrgh"]);
    (Shout, [T "aooogah!"]); (Sentence, [N Quiet]); (Sentence, [N Grunt]);
    (Sentence, [N Shout]); (Conversation, [N Snore]);
    (Conversation, [N Sentence; T ","; N Conversation])])
val giant_test0 : bool = true
val giant_test1 : bool = true
val giant_test2 : bool = true
# #use "hw1test.ml";;
val my_subset_test0 : bool = true
val my_subset_test1 : bool = true
val my_subset_test2 : bool = true
val my_subset_test3 : bool = true
val my_equal_sets_test0 : bool = true
val my_equal_sets_test1 : bool = true
val my_equal_sets_test2 : bool = true
val my_set_union_test0 : bool = true
val my_set_union_test1 : bool = true
val my_set_union_test2 : bool = true
val my_set_intersection_test0 : bool = true
val my_set_intersection_test1 : bool = true
val my_set_intersection_test2 : bool = true
val my_set_diff_test0 : bool = true
val my_set_diff_test1 : bool = true
val my_set_diff_test2 : bool = true
val my_computed_fixed_point_test0 : bool = true
val my_computed_fixed_point_test1 : bool = true
val my_computed_fixed_point_test2 : bool = true
val my_computed_periodic_point_test0 : bool = true
val my_computed_periodic_point_test1 : bool = true
val my_while_away_test0 : bool = true
val my_while_away_test0 : bool = true
val my_rle_decode_test0 : bool = true
val my_rle_decode_test1 : bool = true
type just_college_thoughts =
    Freshmen
  | Sophomore
  | Happiness
  | Sad
  | Depression
  | Expletive
  | Dropout
val college_thoughts_grammar :
  just_college_thoughts *
  (just_college_thoughts * (just_college_thoughts, string) symbol list) list =
```

```
  (Freshmen,
   [(Freshmen, [N Happiness]); (Freshmen, [N Sad]);
    (Freshmen, [N Happiness; N Sad; T "confusion"]);
    (Freshmen, [N Sophomore]); (Sophomore, [N Sophomore]);
    (Sophomore, [N Sad; N Depression]); (Sophomore, [N Depression]);
    (Depression, [N Depression]); (Sad, [N Expletive]);
    (Expletive, [N Dropout; T "welp"]); (Expletive, [T "small bad word"]);
    (Expletive, [T "big bad word"]); (Expletive, [T "many bad word"]);
    (Dropout, [T "such is life"])])
val my_blind_alley_test0 : bool = true
```