```
(*1*)
let rec subset a b = match a with
                [] -> true |
                h::t -> if List.exists (fun x -> x = h) b
                then subset t b else false
                ;;

(*2*)
let equal_sets a b = subset a b && subset b a
;;

(*3*)
let set_union a b = a@b
;;

(*4*)
let rec set_intersection a b=match a with
                               [] -> []|
                               h::t -> if List.exists (fun x -> x = h) b
                                          then h::set_intersection t b
                                          else  set_intersection t b
                                          ;;

(*5*)
let rec set_diff a b= match a with
                        []->[]|
                        h::t -> if List.exists(fun x -> x == h) b
                                  then set_diff t b
                                  else h::set_diff t b
                                  ;;

(*6*)
let rec computed_fixed_point eq f x= if eq (f x) x
                          then x
                          else computed_fixed_point  eq f (f x)
                          ;;

(*7*)
let rec computed_periodic_point eq f p x = if p = 0
                          then x
                          else if eq x (f (computed_periodic_point eq f (p-1) (f x)))
                                 then x
                                 else (computed_periodic_point eq f p (f x))
                                 ;;

(*8*)
let rec while_away s p x = if p x then x :: while_away s p (s x) else []
;;

(*9*)
let rec rle_decode lp = match lp with
            []-> []|
            h::d->let (count, chara)=h
                          in
                          let rec helper a b=match a with
                          0 -> []|
                          _ -> b::helper (a-1) b
                          in
                          (helper count chara)@rle_decode d
                          ;;



type ('nonterminal, 'terminal) symbol =
| N of 'nonterminal
| T of 'terminal
;;
(*为true 如果整个rule能产生termial，为假如果发生无限递归或者找不到*)
let rec find_symbo symbo rule ruleo table = match rule with
[]->true|
 h::t-> match h with
            T d->find_symbo symbo t ruleo table|
            N n->if n = symbo
                      then false
                      else  if find_table n table (List.filter(fun x-> x <>(symbo, ruleo)) table)
```

```
                              then find_symbo symbo t ruleo table
                              else false

and find_table symbo table otable= match table with
 []->false|
 h::t-> match h with
          (nt, rule)->if nt = symbo then
                                        if find_symbo nt rule rule otable
                                        then true
                                        else find_table symbo t otable
                            else find_table symbo t otable
;;

let rec check_list rule table=match rule with
[]->true|
h::d->match h with
          T t->check_list d table|
          N n->if find_table n table table
                  then check_list d table
                  else false
 ;;

 let rec check_grammar table otable= match table with
 []->[]|
 h::t->match h with
         (title, rule)->if check_list rule otable
                              then h::check_grammar t otable
                              else check_grammar t otable
 ;;
let filter_blind_alleys g=match g with
          (h, t)->(h, check_grammar t t)
;;

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"]]
;;
```