Γλώσσες Προγραμματισμού 2
'Ασκηση 10

'Αγγελος Πλεύρης
3115038

October 24, 2020

Αξιωματική Σημασιολογία

Δεδομένα

Μας δίνεται ο παρακάτω κώδικας μια συνάρτησης σε C.

```
#define MAXN 1000000
#define MAXV 2000000

int samenum(int N,int* x) {
  int p[MAXV+1];
  for (int = 0; i<= MAXV; i++) p[i]= -1;
  int best = 0;
  for (int i=0; i < N; i++) {
    if (p[x[i]] == -1) p[x[i]] = i;
    else if (i-p[x[i]] > best) best = i - p[x[i]];
  }
  return best;
}
```

Καλούμαστε να αποδείξουμε ότι το αποτέλεσμα της παραπάνω συνάρτησης $res \neq 0$ αν και μονο αν :

- υπάρχουν x[i],x[j] στοιχεία τέτοια ώστε x[i] = x[j] και $i < j$ και res = j - i.
- Για οποιαδήποτε άλλα στοιχεία x[i'] και x[j'] τέτοια ώστε x[i'] = x[j'] και $i' < j'$ ισχύει $res \geq j' - i'$

Δηλαδή καλούμαστε να αποδείξουμε την παρακάτω σχέση :
(1) : $res \neq 0 \iff \exists i, j; (0 <= i < j \wedge x[i] = x[j] \wedge res = j - i) \wedge (\forall i', j'; 0 <= i' < j' \wedge (i' \neq i \vee j' \neq j) \wedge x[i'] == x[j'] \implies res >= j' - i')$

Απόδειξη

Ευθύ

Για να αποδείξουμε το ευθύ θα αντιστρέψουμε τη συνεπαγωγή και έτσι έχουμε :
$res \neq 0 \implies \exists i, j; (0 <= i < j \wedge x[i] = x[j] \wedge res = j - i) \wedge (\forall i', j'; 0 <= i' < j' \wedge (i' \neq i \vee j' \neq j) \wedge x[i'] == x[j'] \implies res >= j' - i') \iff \neg\exists i, j; (0 <= i < j \wedge x[i] = x[j] \wedge res = j - i) \wedge (\forall i', j'; 0 <= i' < j' \wedge (i' \neq i \vee j' \neq j) \wedge x[i'] == x[j'] \implies res >= j' - i') \implies res = 0 \iff$
Επειδή οι 2 προτάσεις συνδέονται μεταξύ τους με λογικό και το οποίο στην άρνηση θα γίνει or αρκεί να αποδείξουμε ότι :

(2) : $\neg\exists i,j;(0 <= i < j \wedge x[i] = x[j] \wedge res = j - i) \implies res = 0$
δηλαδή ότι αν δεν υπάρχει λύση το αποτέλεσμα της συνάρτησης θα είναι 0. Για να το αποδείξουμε αυτό θα εξετάσουμε τη συνθήκη αυτή στη βασική λούπα της συνάρτησης.
Για i = 0 πριν αρχίσει να τρέχει η λούπα η συνθήκη (2) είναι τετριμμένη και ισχύει. Το χρησιμοποιούμε σαν επαγωγική βάση. Θα θεωρήσουμε ότι για κάποιο k ισχύει η (2) και θα δούμε αν ισχύει και για k+1.
Έστω ότι βρισκόμαστε στην αρχή της k-επανάληψης και η (2) επαληθέυεται. Αν το x[k] δεν είναι ίσο με κανένα από τα προηγούμενα στοιχεία και επειδή ισχύει και η επαγωγική υπόθεση ούτε τα προηγούμενα στοιχεία είναι ίσα μεταξύ τους (δεν υπάρχει έστω και ένα ζευγάρι ίσων στοιχείων), τότε το x[k] θα τρέξει το πρώτο if της βασικής λούπας και έτσι θα έχουμε ότι p[x[k]] = k και άρα δεν θα μεταβληθεί η τιμή του best. Επομένως αφού λόγω επαγωγικής υπόθεσης δεν υπάρχουν στοιχεία i,j ώστε res = j - i, δείξαμε ότι ούτε τώρα θα υπάρχουν τέτοια στοιχεία, δηλαδή θα ισχύει η (2) και στην αρχή της k+1 επανάληψης. Επομένως, αφού ισχύει και η επαγωγική βάση η (2) ισχύει για τη βασική λούπα της συνάρτησης και άρα αποδείξαμε το ευθύ.

Αντίστροφο

Για την απόδειξη του αντίστροφου χρησιμοποιήσαμε το εργαλείο frama-c, με τη χρήση των invariants στις γραμμές 35-53. Παρακάτω, παρουσιάζουμε το predicate που θέλουμε να αποδείξουμε και τα βασικά invariants της απόδειξης.
Την απόδειξη την τρέχουμε με την εντολή :
frama-c -wp -wp-prover alt-ergo -wp-rte -wp-timeout 300 -wp-verbose 0 samenum.c -then -report
και η έκδοση του frama-c που χρησιμοποιήθηκε ήταν η Phosphorus-20170501
.
Predicate :

```
/*@ predicate validres{L}(int n, int res, int *a) =
  @      (\exists integer i,j; 0 <= i < j < n && \at(a[i],L) == \
     at(a[j],L) && res == j - i
              && (\forall integer k,l; 0<= k < l < n && (k != i
                  || l != j) && \at(a[k],L) == \at(a[l],L) ==>
                  res >= l - k)) ==> res != 0;
  @*/
/*@ requires 2 <= N < MAXN;
  @ requires \valid (x + (0..(N-1)));
  @ requires \forall integer i; 0 <= i < N ==> 1 <= x[i] < MAXV;
  @ assigns \nothing;
  @ ensures validres(N,\result,x);
@*/
```

Invariants :

```
/*@ loop invariant !(\exists integer k,l; 0<= k < l < i && x[k]
   == x[l] && best == l- k) ==> (\forall integer k,l; 0<= l < k
   < i ==> x[l] != x[k] || best != l - k) ;

    @ loop invariant \forall integer k; 0<= k < i ==> (\exists
       integer l; 0<= l < k && p[x[k]] == l ==> x[k] == x[l]);

    @ loop invariant \forall integer k; 0<= k < i && k - p[x[k]]
        > 0 ==> best >= k - p[x[k]];

    @ loop invariant \exists integer k,l; 0 <= k < l < i && x[k]
        == x[l] ==> best != \at(best,Pre) && best == l - k;

    @ loop invariant \forall integer k; 0<= k < i ==> (p[x[k]]
       == k ==> \forall integer l; 0<= l < k ==> x[l] != x[k]);

    @ loop invariant \forall integer k; 0 <= k < i ==> \forall
       integer l; 0 <= l < p[x[k]] ==> x[l] != x[k];

    @ loop invariant best == \at(best,Pre) ==> (\forall integer
       k,l; 0<= k < l < i ==> x[l] != x[k]);

    @ loop invariant best != \at(best,Pre) ==> (\forall integer
       k; 0<= k < i && k != p[x[k]] ==> best >= k-p[x[k]]);

    @ loop invariant \forall integer k; 0<= k < i ==> p[x[k]] !=
        k ==> (\exists integer l; 0<= l < k < i && p[x[k]] == l
       ==> x[k] == x[l]);

    @ loop invariant \forall integer k; i <= k < N && p[x[k]] !=
        -1 ==>
                           \exists integer y; 0<= y < i && p[x[k
                              ]] == y;
@*/
```