Teste Frama-C + CBMC (2014/2015)

PARTE 1

Considere o seguinte programa (dispinível em questao1.c) para testar se um dado elemento está presente num array.

```
/*@  requires size>=0 && \valid(arr+(0..(size-1)));
     ensures \result==0 <==> \forall integer k; 0<=k<size ==> arr[k]!=y;
     assigns \nothing;
*/
int belongs (int y, int arr[], int size)
{
  int i=0;
  while (i<size) {
    if (arr[i]==y) return 1;
    i = i+1;
  }
  return 0;
}
```

A. Experimente fazer a verificação deste programa tal como está. Porque razão a prova falha?

B. Acrescente anotações de forma a conseguir fazer a prova das propriedades de **safety**.

C. Complete agora as anotações que forma a conseguir provar a correcção do programa face ao seu contrato.

D. Defina um predicado lógico que modele o funcionamento desta função e reescreva o contrato da função usando esse predicado.

PARTE 2

Considere agora a seguinte função (disponínel em questão2.c) que recebe três arrays A, B e C de dimensões na, nb e nc, repectivamente, e coloca em C os elementos de A que também pertencem a B. A função devolve o número de elementos escritos em C.

```
int diff (int A[], int sizeA, int B[], int sizeB, int C[], int sizeC)
{
  int i, j=0;

  for (i=0; i<sizeA; i++)
    if (belongs(A[i],B,sizeB)) {
      C[j] = A[i];
      j++;
    }

  return j;
}
```

A. Escreva um contrato que caracterize de forma completa o comportamento desta função.

B. Prove a correcção da função em ordem ao contrato que escreveu. No caso de não conseguir completar a prova na totalidade, identifique claramente as VCs que estão a falhar e indique o que já fez para tentar contornar o problema. O que acha que pode estar a acontecer?

C. Analisando a função percebe-se que se o array A não tiver elementos repetidos, podemos garantir que o array C também não terá. Enriqueça o contrato que escreveu com esta informação. (Não precisa de se preocupar em fazer a prova deste novo contrato.)

PARTE 3

Considere agora a seguinte função (dispinível em questao3.c) que recebe um array A de dimensão na e um valor x e substitui no array todos os valores superiores a x por x.

```
void threshold (int arr[], int size, int x)
{
  int i;

  for (i=0; i<size; i++)
    if (arr[i]>x) arr[i] = x;
}
```

A. Escreva um contrato que caracterize de forma completa o comportamento desta função.

B. Prove a correcção da função em ordem ao contrato que escreveu. No caso de não conseguir completar a prova na totalidade, identifique claramente as VCs que estão a falhar e indique o que já fez para tentar contornar o problema. O que acha que pode estar a acontecer?

C. Analisando a função percebe-se que o somatório de todos os elementos dos array no prós-estado deve ser inferior ao somatório dos valores do array no pós-estado. Indique o que necessita de fazer para colocar como pós-condição esta propriedade. (Não precisa de se preocupar em fazer a prova deste novo contrato.)

PARTE 4

Considere o seguinte programa (disponível em questao4.c)

```
#define N 100

int vec[N];

int min(int A[], int size) {
  int i = 1;
  int m = 0;

  while (i < size) {
    if (A[i] < A[m]) { m = i; }
    i++;
  }
  return m;
}

void main() {
  int i;

  for (i=0; i<N; i++) {
    vec[i] = i;
  }

  int r = min(vec, N);
}
```

A. Para verificar a segurança da função minarray relativamente a acessos ilegais a posições do array. Invoque

cbmc questao4.c --function min --bounds-check --pointer-check

Por que razão não pára a execução?

B. Repita com a invocação com

cbmc --function min --unwind 100 --bounds-check --pointer-check

Por que razão falha agora a verificação?

Será que podemos evitar esta falha?

O que podemos dizer acerca da função min relativamente a acessos ilegais a posições do array?

C. Repita com a invocação com

cbmc questao4.c --bounds-check --pointer-check

O que podemos dizer acerca da segurança do programa main?

D. Insira uma asserção que permita verificar que o resultado da função min é um índice legal do array.

E. Insira código que permita verificar que o conteúdo da posição r calculada por min é não superior ao conteúdo de qualquer outra posição do array.

PARTE 5

Considere o seguinte programa (disponível em questao5.c)

```
#define N 15

int sum (int V[], int N) {
  int i, s = 0;

  for (i=0; i<N; i++)
    s = s + V[i];

  return s;
}

void main() {
  int i, y;
  int arr[N] = {33,25,-43,1,5,3,-2,4,-5,19,0,23,11,53,1} ;

  y=12;

  int s = sum (arr, y);
}
```

A. Use o CBMC para verificar a segurança da função somatorio relativamente a problemas de overflow. O que podemos dizer sobre isso?

B. Use agora o CBMC para verificar a segurança do programa principal quanto a problemas de overflow e eventuais acessos ilegais ao array. A que conclusão chegou?

C. Substitua agora a atribuição inicial ao array arr por uma atribuição não determinista. Insira código que lhe permita fazer isso e volte a testar a segurança do programa principal quanto a problemas de overflow. A que conclusão chegou?

D. Queremos agora simular que o valor da variável y pode variar dentro da gama de indices válidos para o array arr (em vez de estar fixo no valor 12). Faça essa simulação usando os mecanismos que o CBMC disponibiliza, e teste a segurança do programa relativamente a eventuais acessos ilegais ao array. A que conclusão chegou?