

Case study: use of separation analysis in proving advanced behavioral property of a C program

Claude Marché

November 2006

1 Problem

The following C program is inspired from a piece of Java code (<http://www.cs.ru.nl/~woj/esfws06/slides/Peter.pdf>) by Peter Müller (ETH Zürich) during the Exploratory Workshop: Challenges in Java Program Verification (<http://www.cs.ru.nl/~woj/esfws06/>). It computes the set of positive elements of an array, and puts them in a new array.

The C source code is the following:

```
int *f(int t[], int length) {
    int count = 0;
    int i;
    int *u;

    for (i=0 ; i < length; i++) {
        if (t[i] > 0) count++;
    }
    u = (int*)calloc(count, sizeof(int));
    count = 0;

    for (i=0 ; i < length; i++) {
        if (t[i] > 0) u[count++] = t[i];
    }
}

return u;
}
```

We assume that the parameters of f satisfy the following properties:

- $\text{length} \geq 0$
- array t is allocated, at least on the interval $[0..\text{length} - 1]$

The point is to show the absence of threat on memory safety, that is there are no access outside the array bounds.

2 Solution using the Caduceus tool

We annotate the source code using the (JML-style) annotation language of Caduceus (<http://caduceus.lri.fr>).

We first introduce a logical function to count positive elements of an array:

```
//@ logic int num_of_pos(int i,int j,int t[]) reads t[i]

/*@ axiom num_of_pos_empty :
  @   \forall int i, int j, int t[];
  @     i > j => num_of_pos(i,j,t) == 0
  @*/

/*@ axiom num_of_pos_true_case :
  @   \forall int i, int j, int k, int t[];
  @     i <= j && t[j] > 0 =>
  @       num_of_pos(i,j,t) == num_of_pos(i,j-1,t) + 1
  @*/

/*@ axiom num_of_pos_false_case :
  @   \forall int i, int j, int k, int t[];
  @     i <= j && ! (t[j] > 0) =>
  @       num_of_pos(i,j,t) == num_of_pos(i,j-1,t)
  @*/

/*@ axiom num_of_pos_strictly_increasing:
  @   \forall int i, int j, int k, int l, int t[];
  @     j < k && k <= l && t[k] > 0 =>
  @       num_of_pos(i,j,t) < num_of_pos(i,l,t)
  @*/
```

The source code is then annotated as follows:

```
/*@ requires length >= 0 && \valid_range(t,0,length-1)
  @*/
int *f(int t[], int length) {
  int count = 0;
  int i;
  int *u;

 /*@ invariant
  @   0 <= i && i <= length &&
  @   0 <= count && count <= i &&
  @   count == num_of_pos(0,i-1,t)
  @ variant length - i
  @*/
  for (i=0 ; i < length; i++) {
```

```

    if (t[i] > 0) count++;
}
u = (int *)calloc(count, sizeof(int));
count = 0;

/*@ invariant
@   0 <= i && i <= length &&
@   0 <= count && count <= i &&
@   count == num_of_pos(0, i-1, t)
@ variant length - i
@*/
for (i=0 ; i < length; i++) {
    if (t[i] > 0) {
        u[count] = t[i];
        count++;
    }
}
return u;
}

```

3 Proof and comments

- 12 Verification Conditions are generated byr Caduceus
- All of them are proved by the Simplify automatic theorem prover
- Crucial condition: activate separation analysis (option `-separation`) of Caduceus