Frama-c Demo with Simple Examples

19CSE205 : PROGRAM REASONING

Dr. Swaminathan J

Assistant Professor

Department of Computer Science and Engineering

AMRITA VISHWA VIDYAPEETHAM

Jul - Dec 2020

Contents

1. Checking max function

The max function computes the bigger of two integers.

File: max.c

```
/*@
     ensures \result == a && \result > b
          || \result > a && \result == b
          || \result == a && \result == b;
*/
int max(int a, int b) {
     return a > b ? a : b;
}
```

Running frama-c on max.c

```
prompt> frama-c -wp max.c
[kernel] Parsing max.c (with
preprocessing)
[wp] warning: Missing RTE guards
[wp] 1 goal scheduled
[wp] Proved goals: 1 / 1
     Qed: 1 (5ms)
prompt>
```

1. Making a mistake in the implementation.
   - return a > b ? a : b+1;
   - frama-c reports that the goal cannot be proved. Rightly so.
2. Making a mistake in correctness criteria.
   - Remove || \result == a && \result == b
   - frama-c reports that the goal cannot be proved. Rightly so.

Degree of correctness in specification.

File: max.c

```
/*@
    ensures \result >= a && \result >= b;
*/
int max(int a, int b) {
    return a > b ? a : b+1;
}
```

```
prompt> frama-c -wp max.c
[kernel] Parsing max.c (with
preprocessing)
[wp] warning: Missing RTE guards
[wp] 1 goal scheduled
[wp] Proved goals: 1 / 1
    Qed: 1 (5ms)
```

- The above correctness criteria is not sufficient to catch the error in the implementation.
  - return a > b ? a : b+1;
  - frama-c reports that the goal is proved. Wrong!
- This is because the degree of correctness as embodied by the criteria is not strong enough.

Runtime considerations associated with semantics.

File: next.c

```
/*@
    requires a > 0;
    ensures \result > 1;
*/
int next(int a) {
    return a + 1;
}
```

```
prompt> frama-c -wp -wp-rte next.c
[kernel] Parsing next.c (with preprocessing)
[rte] annotating function next
[wp] 2 goals scheduled
[wp] [Alt-Ergo] Goal
typed_next_assert_rte_signed_overflow :
Unknown (Qed:2ms) (54ms)
[wp] Proved goals: 1 / 2
    Qed: 1
    Alt-Ergo: 0 (unknown: 1)
```

- Although the logic is in line with correctness specification, a bug is lurking from "runtime semantics" standpoint.
  - For any a >= 2147483647, output is incorrect due to integer overflow.
- If the precondition is specified the following way, both static and runtime goals will be proved.
  - requires a > 0 && a < 2147483647;

The problem with pass-by-value approach.

File: swap.c

```
/*@ ensures a == \old(b)
          && b == \old(a);
*/
int swap(int a, int b) {
      int temp = a;
      a = b;
      b = temp;
}
```

```
prompt> frama-c -wp swap.c
[kernel] Parsing swap.c (with preprocessing)
[wp] warning: Missing RTE guards
[wp] 1 goal scheduled
[wp] [Alt-Ergo] Goal typed_swap_post :
Unknown (Qed:2ms) (505ms)
[wp] Proved goals: 0 / 1
     Alt-Ergo: 0 (unknown: 1)
```

- Swapping of local variables lasts only till the function lasts.
  - a and b inside swap function are different from a and b outside.
  - They refer to different memory locations.
- If a and b are made global variables, then the goal will be proved.
  - Declare int a, b; above annotated comments.
  - Remove arguments a and b from swap definition.

The advantage of pass-by-pointer approach.

File: swap.c

```
/*@ ensures *a == \old(*b)
          && *b == \old(*a);
*/
int swap(int *a, int *b) {
      int temp = *a;
      *a = *b;
      *b = temp;
}
```

```
prompt> frama-c -wp swap.c
[kernel] Parsing swap.c (with preprocessing)
[wp] warning: Missing RTE guards
[wp] 1 goal scheduled
[wp] Proved goals: 1 / 1
     Qed: 0 (6ms)
     Alt-Ergo: 1 (13ms) (17)
```

- Now the postcondition is satisfied. The goal is proved.
  - Although a and b inside swap are different from a and b outside, they point to the same respective locations when de-referenced.
- However, using -wp-rte option shows 4 goals cannot be proved.
  - Because a and b are de-referenced at 4 points within the function. What if caller of swap passed null pointers? Segmentation fault!
  - To be safe, include requires \valid(a) && \valid(b) to contract.

6. Checking time counter

AMRITA VISHWA VIDYAPEETHAM

File: counter.c

```
struct Counter {
      int seconds;
};
struct Counter c; // global

/*@
        behavior one:
          assumes 0 <= c.seconds < 59;
          ensures c.seconds == \old(c.seconds)+1;
        behavior two:
          assumes c.seconds == 59;
          ensures c.seconds == 0;
*/
void tick() {
      c.seconds = (c.seconds+1) % 60;
}
```

```
prompt> frama-c -wp counter.c
[kernel] Parsing counter.c (with preprocessing)
[wp] warning: Missing RTE guards
[wp] 1 goal scheduled
[wp] Proved goals: 2 / 2
      Qed: 1 (0.25ms-4ms)
      Alt-Ergo: 1 (12ms) (15)
```

Try the following exercises.

1. Run with -wp-rte option.
   - What problem do you observe?
   - Why does the problem occur?
   - How do you fix the problem?
2. Include minute and hour variables to Counter structure.
   - Rewrite the code to handle updates to minute and hour as per their ranges.
   - Add new behaviors and make the specification complete.
3. Try to specify complete and disjoint behaviors as appropriate.

What to focus?

While learning Frama-c from references, keep the following in mind.

- Don't spend your time learning more and more features of frama-c.
- Instead focus on learning how to specify correctness criteria. It demands original thinking and that is the key skill to acquire.
- While learning more syntactical features can be of some help, keep in mind that frama-c will not think for you.
- Do exercises on your own. It is far better to try and not get the criteria right than borrowing someone else's solution. Atleast your thinking ability would have got stretched.
- Seek help only after you have tried enough and reached a dead-end.
- If you have got the criteria right and someone is asking your help, give him/her hints instead of sharing your solution.