Verifying Algorithms. Notes.

Ilgiz Mustafin

Innopolis University

August 16, 2017

1 Installing EVE (0:37)

Since I am using Linux, I took the latest linux build (96960) available at that time.

Unpacking and running `run_eve.py` worked (Account example could be verified as expected).

Off topic: I am using a dark GTK theme (Breeze-Dark) on my computer. EiffelStudio has some problems with this (Light grey text on white background, etc.), so I start it with light theme with `GTK2_RC_FILES=/usr/share/themes/Breeze/gtk-3.20/gtkrc python run_eve.py`.

2 Tutorial

Before I could start working on algorithms, I had to complete the tutorial from `http://se.inf.ethz.ch/research/autoproof/tutorial/`.

2.1 Exercise 1 (1:07)

In feature `increase_seconds` pre-condition `seconds = 59 implies modify_model ("minutes", Current)` gives "Manifest string not supported" and "Manifest tuple in contract not supported" errors, while just `modify_model("minutes",`

`Current`) works. It seems that `modify_model` is not supported in logical expressions. Also, in solutions there is no such check, there is just `modify_model (["seconds", "minutes", "hours"], Current)`.

2.2 Exercise 2 (3:40)

2.2.1 Binary Search

When doing the Binary Search task, I was required to write a precondition to check that array is sorted.

My code was:

```
across 1 |..| (a.sequence.count - 1) as c all
    a.sequence[c.item] <= a.sequence[c.item + 1]
end
```

While the code from the solutions was:

```
across 1 |..| a.sequence.count as i all
across 1 |..| a.sequence.count as j all
    i.item <= j.item implies
        a.sequence[i.item] <= a.sequence[j.item]
end
end
```

These two variants give different results of verification. I was able to verify the function with the code from the solutions, but not with my code (`not_in_lower_part` and `not_in_upper_part` could be violated).

This could be so, if AutoProof was not considering that $\leq$ is transitive for integers. Maybe worth pointing out in tutorial.

2.2.2 Permutations

For the Permutations task there was a feature for testing different permutation checking algorithms. I added some more tests.

```
feature test_permutation
local
    s1, s2: MML_SEQUENCE [INTEGER]
do
    s1 := << 1, 2, 3, 4 >>
    s2 := << 1 >>
    check p1: not is_permutation_1 (s1, s2) end
```

```
8
9       s1 := << 1, 2, 3, 4 >>
10      s2 := << 1 >>
11      check p2: not is_permutation_2 (s1, s2) end
12
13      s1 := << 1, 2, 2 >>
14      s2 := << 1, 1, 2 >>
15      check p2_2: not is_permutation_2 (s1, s2) end
16
17      s1 := << 1, 2, 3, 4 >>
18      s2 := << 1 >>
19      check p3: not is_permutation_3 (s1, s2) end
20
21      s1 := << 1, 2, 2 >>
22      s2 := << 1, 1, 2 >>
23      check p3_2: not is_permutation_3 (s1, s2) end
24
25      s1 := << 1, 2, 3, 4 >>
26      s2 := << 1 >>
27      check p4: not is_permutation_4 (s1, s2) end
28
29      s1 := << 1, 2, 2 >>
30      s2 := << 1, 1, 2 >>
31      check p4_2: not is_permutation_4 (s1, s2) end
32  end
```

In this version AutoProof reports that checks `p1` and `p2_2` may be violated, but check `p4_2` should fail too.

If we comment check `p1`, only `p2` is reported.

If we comment check `p2`, then both `p1` and `p4_2` are reported.

It seems that when AutoProof finds some violation in a feature, it stops checking that feature and several (but *not all!*) violations will be reported. This can be very confusing for people unfamiliar with such behavior.

2.3 Excercise 3 (2:17)

Nothing interesting.

3 Real development

3.1 Generics, `int` and `ref`

Suppose we have class `VALUE` which stores one non-void object.

```
class
    VALUE [G]

create
    make

feature

    make (v: G)
        require
            value_exists: v /= Void
        do
            value := v
        end

    value: G

end
```

We want to use it to store an integer.

```
class
    TEST

feature

    test
        local
            value: VALUE[INTEGER]
        do
            create value.make (123)
        end

end
```

Trying to verify this system gives `invalid argument types (int and ref) to binary operator !=`.

After quick inspection of AutoProof `*.bpl` output in `EIFGENs/.../Proofs` I suspect several facts.

```
type ref; // Type definition for reference types
const Void: ref; // Constant for Void references
```

Eiffel `Void` is translated into a Boogie constant of reference type.

Eiffel `INTEGER` is translated into Boogie `int`. There is some code about integer boxing, but both these functions are never used.

```
const unique INTEGER: Type;
function boxed_int(i: int) returns (ref);
function unboxed_int(r: ref) returns (int);
```

These suspicions are supported by the following code.

```
procedure VALUE^ANY^.make(Current: ref, v: ref);
  ...
  requires (v) != (Void); // type:pre tag:v_exists line:11
  ...
...
procedure VALUE^INTEGER_32^.make(Current: ref, v: int);
  ...
  free requires is_integer_32(v); // info:type property for argument v
  ...
  requires (v) != (Void); // type:pre tag:v_exists line:11
  ...
```

While the assertion `(v) != (Void)` is valid in the `ANY` variant because argument `v` is of type `ref`, the code in the `INTEGER` is invalid because `v` is of type `int`.

The first naive solution was to add an axiom `axiom (forall i:int :: (Void) != i);` to the theory files, but this is not even a valid Boogie construction (invalid argument types (ref and int) to binary operator `!=`).

For now, I will make less generic code as a workaround for this problem.

3.2 Using object comparison to find objects in MML containers, `is_equal` vs `is_model_equal`

Suppose we have class `TS_PAIR` which is just a pair of integers.

```
note
    model: left, right

class
    TS_PAIR

inherit

    ANY
```

```
10          redefine
11              is_equal,
12              is_model_equal
13          end
14
15  create
16      make
17
18  feature -- Initialization
19
20      make (a_left, a_right: INTEGER)
21          note
22              status: creator
23          require
24              modify_model (["left", "right"], Current)
25          do
26              left := a_left
27              right := a_right
28          ensure
29              left = a_left
30              right = a_right
31          end
32
33  feature -- Data
34
35      left: INTEGER assign set_left
36
37      right: INTEGER assign set_right
38
39  feature -- Modification
40
41      set_left (a_left: INTEGER)
42          require
43              modify_model ("left", Current)
44          do
45              left := a_left
46          ensure
47              left = a_left
48          end
49
50      set_right (a_right: INTEGER)
51          require
52              modify_model ("right", Current)
53          do
54              right := a_right
55          ensure
56              right = a_right
57          end
58
```

```
feature -- Comparison

    is_equal, is_model_equal (a_other: like Current): BOOLEAN
        note
            status: functional
        require else
            reads_model (["left", "right"], [Current, a_other])
        do
            Result := left = a_other.left and right = a_other.right
        end

end
```

How to check if there is already an object-equal pair in a, say, `MML_SEQUENCE`? This code will not work (Check `s_has_object_b` may be violated.):

```
local
    a, b: TS_PAIR
    s: MML_SEQUENCE [TS_PAIR]
do
    create a.make (1, 1)
    create b.make (1, 1)
    create s
    s := s & a
    check
        s_has_object_b: across s as i some i.item.is_equal (b)  end
    end
end
```

While trying to find the root of the problem I added some random checks which made verification successful.

```
local
    a, b: TS_PAIR
    s: MML_SEQUENCE [TS_PAIR]
do
    create a.make (1, 1)
    create b.make (1, 1)
    create s
    s := s & a
    check
        a_equals_b: a.is_equal (b)
    end
    check
        s_has_a: s.has (a)
    end
    check
        s_has_object_b: across s as i some i.item.is_equal (b)  end
    end
end
```

Changing `a_equals_b` check into `b.is_equal (a)` invalidates `s_has_object_b`. This makes sense becuase `ANY.is_equal` does not have `symmetric` contract as `ANY.is_model_equal` does.

Using `is_model_equal` instead of `is_equal` requires `a_equals_b` and `s_has_a` checks to verify `s_has_object_b`, but we can swap left and right sides in `a_equals_b`.

What function should I use in my code? Let us compare contracts.

```
is_model_equal (other: like Current): BOOLEAN
        -- Is the abstract state of `Current' equal to that of `other'?
    require
            other /= Void
            reads (Current, other)
    ensure
        reflexive: other = Current implies Result
        symmetric: Result = other.is_model_equal (Current)
```

```
is_equal (other: like Current): BOOLEAN
        -- Is `other' attached to an object considered
        -- equal to current object?
    require
        other_not_void: other /= Void
```

We can see that `is_equal` has neither reflexivity nor symmetricity properties, which are natural properties to assume while designing code.

`is_model_equal`, on the other hand, should only be used in contracts for verification. Normal Eiffel class `ANY` does not have this function.

We could try to mix them like this:

```
if x.is_equal(y) then -- for behavior, we may assume symmetricity, etc.
    check x.is_model_equal (y) end -- Verify symmetricity, etc.
    ...
```

But there is no guarantee that for all `x` and `y` `x.is_equal (y) = x.is_model_equal (y)`. Adding such contract can be a solution.

For the first versions of algorithm I will use `is_model_equal` in custom classes, until we come up with something better.

3.3 Instantiation of `V_ARRAY2`

```
items: INTEGER

data: V_ARRAY2[BOOLEAN]
```

```
4
5   make (a_items: INTEGER)
6       require
7           a_items >= 0
8       do
9           items := a_items
10          create data.make_filled (a_items, a_items, False)
11          check rows: data.row_count = items end
12          check columns: data.column_count = items end
13      end
```

First of all, check `rows` could not be verified, but check `columns` is verified. Maybe this is because `V_ARRAY2#make` and `#make_filled` have a post-condition that `#column_count` is set, but they do not have a post-condition that `#row_count` is set. Maybe this is because it can be derived from invariant `row_count_definition: row_count * column_count = sequence.count`.

4 Installation 2

I had to switch to a new computer. This required me to install EVE to a fresh system. I was using ArchLinux. I did the same installation steps: download the freshest EVE build for linux, unpack the archive, run EVE.

However, issuing AutoProof verification from EVE produced no output in EVE, `EIFGENs/v_topsort/Proofs/autoproof*.txt` files were created but were empty. `bpl` files are created and contain Boogie code, they seem ok. Nothing is written to STDOUT/STDERR of EVE.

Following EVE installation instructions from `https://trac.inf.ethz.ch/trac/meyer/eve/#Setup1` is not working too. Executing `python eve.py update` produces errors (`python2` is for running python 2):

```
$ python2 eve.py update
Updating EiffelStudio
Traceback (most recent call last):
  File "eve.py", line 1196, in <module>
    main()
  File "eve.py", line 1161, in main
    update_EiffelStudio()
  File "eve.py", line 591, in update_EiffelStudio
    name, filename, version, url = get_nightly_build(d_ise_platform, d_archive_e
  File "eve.py", line 614, in get_nightly_build
```

```
    response = urllib2.urlopen(download_page)
  File "/usr/lib/python2.7/urllib2.py", line 154, in urlopen
    return opener.open(url, data, timeout)
  File "/usr/lib/python2.7/urllib2.py", line 429, in open
    response = self._open(req, data)
  File "/usr/lib/python2.7/urllib2.py", line 447, in _open
    '_open', req)
  File "/usr/lib/python2.7/urllib2.py", line 407, in _call_chain
    result = func(*args)
  File "/usr/lib/python2.7/urllib2.py", line 1413, in ftp_open
    fw = self.connect_ftp(user, passwd, host, port, dirs, req.timeout)
  File "/usr/lib/python2.7/urllib2.py", line 1435, in connect_ftp
    persistent=False)
  File "/usr/lib/python2.7/urllib.py", line 877, in __init__
    self.init()
  File "/usr/lib/python2.7/urllib.py", line 889, in init
    self.ftp.cwd(_target)
  File "/usr/lib/python2.7/ftplib.py", line 562, in cwd
    return self.voidcmd(cmd)
  File "/usr/lib/python2.7/ftplib.py", line 254, in voidcmd
    return self.voidresp()
  File "/usr/lib/python2.7/ftplib.py", line 229, in voidresp
    resp = self.getresp()
  File "/usr/lib/python2.7/ftplib.py", line 224, in getresp
    raise error_perm, resp
URLError: <urlopen error ftp error: 550 Failed to change directory.>
```

However, the installation instructions mentioned that *Mono* is required. I installed it and AutoProof started to work.

5 Progress!

Nearly 40 hours were required to get started. I am using *explicit wrapping, contracts* so that I am really understanding what is happening. Making AutoProof display collaborated code could be really helpful.

6 Left-right scrolling

Using touchpad to scroll up and down works, using touchpad to scroll left and right does not. It works in other applications like Firefox.

7 Changing behavior for easier verification

Let us look at an implementation of `is_subset_of`.

```
is_subset_of (a_other: TS_INTEGER_RELATION): BOOLEAN
require
    closed
    a_other.closed
local
    i: INTEGER
    k: INTEGER
do
    Result := True
    from
        i := 1
    invariant
        Result implies across 1 |..| (i - 1) as j all a_other.has_pair
        ↪  (data.sequence[j.item]) end
        Result implies k = 0
        (not Result) implies across 1 |..| (i - 1) as j some not
        ↪  a_other.has_pair (data.sequence[j.item]) end
        (not Result) implies (1 <= k and k <= data.sequence.count)
        (not Result) implies not a_other.has_pair (data.sequence[k])
    variant
        data.sequence.count - i
    until
        i > data.count or not Result
    loop
        if not a_other.has_pair (data[i]) then
            check not a_other.has_pair (data.sequence[i]) end
            Result := False
            k := i
        end
        i := i + 1
    end
ensure
    closed
    a_other.closed
    is_empty implies Result
    a_other.is_empty implies (Result = is_empty)
```

```
35      Result implies across data.sequence as s all a_other.has_pair (s.item)
        ↪  end
36      (not Result) implies across 1 |..| (data.sequence.count) as j some not
        ↪  a_other.has_pair (data.sequence[j.item]) end
37  end
```

In this example, variable `k` is not needed to compute the result of this function. But it helps to verify this function. All usages of `k` seem to be extractable into some ghost function, but I did not do this.

8 Studying AutoProof Verified Code Repository

In some paper I found the link to `http://se.inf.ethz.ch/research/autoproof/repo/index.html`. Here are some comments.

8.1 Weak specifications

Many of the example specifications seem to be too weak. This fact makes these examples somewhat useless. In particular:

sum_and_max Feature `sum_and_max`, according to the comment description of the feature should find the sum of elements in array and find the greatest value in array. However, the postcondition and loop invariants do not even check that `max` is a value present in the array. For any array tuple `[sum = X, max = X*LENGTH]` will be a valid result for postcondition.

master_clock Feature `tick` in class `MASTER` has comment *Increment time*, however postcondition is just `time > old time` but strangely not `time = old time + 1`.

8.2 Pattern to own container and everything in it

Suppose we want an object to own a container and all items in it.

First idea would be to write `invariant owns = container.sequence.range.extended (container) end`. This will not work. AutoProof will say that on this line `container.sequence` might not be readable.

What I see in the Repository and what I randomly did once is to add `owns.has (container)`. So, one of the correct codes to achieve this is this:

```
1 invariant
2     owns.has (container)
3     owns = container.sequence.range.extended (container)
4 end
```

8.3 Using ghost attributes

Due to the fact that I found very few examples of using AutoProof with Eiffel it was really unclear to me how to properly use ghost attributes. The most common usage is assigning a value in invariant like `owns = [a, b, c]`.

However, in `treemax` example we can see another usage: ghost attribute `sequence` is assigned values in creator procedures.

8.3.1 Ghost result for non-ghost functions

In `map` example functions `extend` (add key-value to map) and `remove` (remove key from map) have comment *Returns index of [removed] key 'k' (ghost value)*. While the returned value should be useless for the clients of the class, it is used in the post-condition of these functions.