Tracing back for clients and tool developers

Bruce M. Adcock
Department of Computer Science & Engineering
The Ohio State University
Columbus OH 43210
Email: adcock.14@osu.edu

***Abstract*—Part of our work going forward is to examine the issue of when a verification condition (VC) is not found to be valid by one of our automated tools. Is the VC valid, but our theory inadequate? Is the VC invalid? If so, is it an issue with the code or the annotated specifications? Are the automated tools correct? For the case of an invalid VC, it has been suggested that we provide a counterexample. Here we will discuss some of the issues of providing such a counterexample, and whether or not its usefulness is limited to invalid VCs.**

In our current state of affairs, VCs are generated to verify that at a particular state, through a particular path, a requirement is met. Therefore, if a VC is not able to be proven automatically, there is already a large amount of information known, such as the necessary path to have the issue, the state the issue occurs, what the VC was generated for, and why. The last piece of information is any output of the automated prover when it was unable to solve the VC

There are many reasons we would still like to generate a counterexample if a VC cannot be proven valid, as opposed to just merely the output of the automated prover. The most important reason is to provide the programmer with concrete feedback as to what can go wrong. Programmers are not intended to be masters at using the automated provers, and so the programmer should be shielded from the output of those tools.

To examine how a counterexample would be generated, consider the case of when the tool SplitDecision is unable to say a VC is valid. At each stage of processing, SplitDecision works to ensure that the altered form is equivalid, i.e., that it is as equally valid as the original VC, but may be stated using different connectives or even variables. The final output, if not simply just "valid" or "invalid", is intended to be of a simpler form than the originally stated problem.

It is of this output that it makes sense to try and find a counterexample, as it is, again, of a simpler form. In order to state this in terms of the original problem, that counterexample must now be walked backwards through the steps done by SplitDecision, so that the counterexample is in terms of variables in the original VC, and therefore more relevant to the programmer.

But what does a counterexample show, exactly? We have three possibilities, in fact. The first is that the VC demonstrates a bug in the code. The second that there is a bug in the annotations provided in the code. The last, and more concerning, is that there is a bug in the prover itself.

Consider the code in Figure 3, which is an attempt at an implementation of the enhancement Contains for ArrayOfUnboundedIntegerAsString, demonstrated in Figures 2 and 1, respectively.

There are 22 VCs generated to prove this code, of which, SplitDecision is unable to prove 2 of them. The first that

```
contract ArrayOfUnboundedIntegerAsStringFacility

  uses UnboundedIntegerFacility

  math subtype ARRAY_MODEL is (
        lb: integer,
        ub: integer,
        s: string of integer
    )
    exemplar a
    constraint
      |a.s| = a.ub - a.lb + 1

  type ArrayOfInteger is modeled by ARRAY_MODEL
    exemplar a
    initialization ensures
        a = (1, 0, empty_string)

    procedure SwapItem (updates a: ArrayOfInteger,
                        restores i: Integer,
                        updates x: Integer)
    requires
        a.lb <= i  and  i <= a.ub
    ensures
        a.lb = #a.lb  and
        a.ub = #a.ub  and
        substring (a.s, 0, i - a.lb - 1) =
            substring (#a.s, 0, i - #a.lb - 1)  and
        substring (a.s, i - a.lb + 1, |a.s|) =
             substring (#a.s, i - #a.lb + 1, |#a.s|)  and
        substring (a.s, i - a.lb, i - a.lb + 1) = <#x>  and
        substring (#a.s, i - #a.lb, i - #a.lb + 1) = <x>

end ArrayOfUnboundedIntegerAsStringFacility
```

Fig. 1: Relevent specifications for ArrayOfUnboundedIntegerAsString

```
contract Contains enhances ArrayOfUnboundedIntegerAsStringFacility

    function Contains (restores a: ArrayOfInteger,
                       restores x: Integer): control
        ensures
            Contains = x is in elements(a.s)

end Contains
```

Fig. 2: Contract for Contains for ArrayOfUnboundedIntegerAsString

```
realization LinearIterative
    implements Contains
    for ArrayOfUnboundedIntegerAsStringFacility

    function Contains (restores a: ArrayOfInteger,
                       restores x: Integer): control

        variable pos: Integer
        variable ub: Integer

        pos := LowerBound (a)
        ub := UpperBound (a)

        loop
            maintains
                a = #a  and
                x = #x  and
                a.lb <= pos  and
                pos <= ub + 1  and
                ub = a.ub  and
                Contains = (x is in
                    elements(substring (a.s, 0, pos - a.lb)))
            decreases
                ub - pos + 1
        while not Contains and not IsGreater (pos, ub) do
            variable y: Integer
            SwapItem (a, pos, y)
            Contains := AreEqual (x, y)
            SwapItem (a, pos, y)
            Increment (pos)
        end loop

    end Contains

end LinearIterative
```

Fig. 3: LinearIterative implementation of Contains for ArrayOfUnboundedIntegerAsString

SplitDecision is unable to prove is shown in Table I.[1] The equivalid VC generated by SplitDecision can be seen in Table II. To say that the processed VC is simpler would be an understatement. It is, however, unclear what the connection is with the original VC. Indeed, the only common variable between the two is x_0! The processed VC has an easy counterexample (any instance where the hypotheses are true and $c_7 = \Lambda$ will do), but what does that mean in this case?

This problem is further compounded by the exploration that SplitDecision must do to make a VC that is this much simpler. Figure 4 shows the number of simultaneous possibly valid (until shown otherwise) restrictions on a counterexample that SplitDecision is handling at each stage. At it's worst, there are 392 different cases at one time. It is this kind of problem that makes tracing back the counterexample automatically vitally important for someone programming in RESOLVE.

There is an additional problem. Ideally, an automated tool like SplitDecision would generate a proof certificate, so that the proof can be independently verified. At the moment, SplitDecision certainly does not generate any such certificate, and, as its development is very fast paced, it is unlikely it would be able to always do so whatever rule is applied by it. For a developer of an automated tool, it is also important to be able to trace a generated counterexample to examine the proof steps if there is an error.

[1]This VC is to prove part of the loop invariant on completion of an interation of the loop in the code.

TABLE I: A VC, unproven

$$
\begin{array}{ll}
 & |a_9.s| = 1 + a_0.ub - a_0.lb \\
\wedge & |a_7.s| = 1 + a_0.ub - a_0.lb \\
\wedge & |a_0.s| = 1 + a_0.ub - a_0.lb \\
\wedge & \langle y_9 \rangle = \text{substring}(a_7.s, pos_5 - a_0.lb, 1 + pos_5 - a_0.lb) \\
\wedge & \langle y_7 \rangle = \text{substring}(a_9.s, pos_5 - a_0.lb, 1 + pos_5 - a_0.lb) \\
\wedge & \langle y_7 \rangle = \text{substring}(a_0.s, pos_5 - a_0.lb, 1 + pos_5 - a_0.lb) \\
\wedge & \langle 0 \rangle = \text{substring}(a_7.s, pos_5 - a_0.lb, 1 + pos_5 - a_0.lb) \\
\wedge & \text{substring}(a_9.s, 1 + pos_5 - a_0.lb, |a_9.s|) = \\
 & \text{substring}(a_7.s, 1 + pos_5 - a_0.lb, |a_7.s|) \\
\wedge & \text{substring}(a_9.s, 0, (pos_5 - a_0.lb) - 1) = \\
 & \text{substring}(a_7.s, 0, (pos_5 - a_0.lb) - 1) \\
\wedge & \text{substring}(a_7.s, 1 + pos_5 - a_0.lb, |a_7.s|) = \\
 & \text{substring}(a_0.s, 1 + pos_5 - a_0.lb, |a_0.s|) \\
\wedge & \text{substring}(a_7.s, 0, (pos_5 - a_0.lb) - 1) = \\
 & \text{substring}(a_0.s, 0, (pos_5 - a_0.lb) - 1) \\
\wedge & pos_5 \not> a_0.ub \\
\wedge & x_0 \notin \text{elements}(\text{substring}(a_0.s, 0, pos_5 - a_0.lb)) \\
\wedge & pos_5 \leq 1 + a_0.ub \\
\wedge & a_0.lb \leq pos_5 \\
\wedge & 1 + a_0.ub - pos_5 \geq 0 \\
\hline
\Rightarrow & a_9.s = a_0.s \\
\wedge & a_0.ub = a_0.ub \\
\wedge & a_0.lb = a_0.lb
\end{array}
$$

TABLE II: Unproven VC, after SplitDecision

$$
\begin{array}{ll}
 & |b_{10}| = 1 + 1 \\
\wedge & |b_8| = 1 + 1 \\
\wedge & 0 \leq 1 + |c_2| \\
\wedge & 0 \leq 1 + |c_7| + |c_2| \\
\wedge & 1 < |b_{10}| \\
\wedge & 1 < |b_8| \\
\wedge & 0 < |b_{10}| \\
\wedge & x_0 \neq d_1 \\
\wedge & e_1 \neq d_1 \\
\hline
\Rightarrow & x_0 \in \text{elements}(c_7)
\end{array}
$$

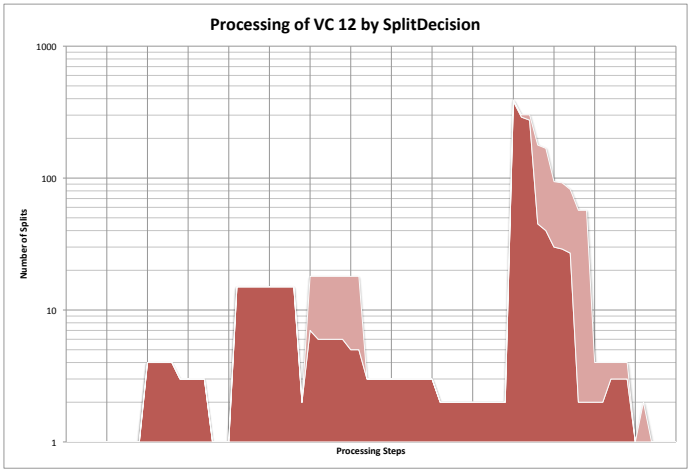

Fig. 4: Number of simultaneous splits handled by SplitDecision at each stage. Vertical scale is logarithmic to be discernible. The difference of color in the graph shows where the final counterexample lay amongst the splits.