User Experiment for Specification Construction Using Behaviors, Equivalences, and SMT Solvers

Paul C. Attie and Fadi Zaraket

American University of Beirut
Email: {pa07,fz11}@aub.edu.lb

You will first login to the server with the account assigned to you using ssh.

```
ssh spec1@192.168.200.6
```

Use the password assigned to you once prompted.

```
spec1pwd
```

Once logged in make a directory with your name and change to it.

```
mkdir YourName
cd YourName
```

Extract the files from the tool compressed folder to your directory.

```
tar -xzf /home/fadi/projects/bb/scma.ship.latest.tgz
```

Now change directory to scma.ship.

```
cd scma.ship
```

Now you need to run the tool using the run theory script, try it with the following flags to write a specification denoting whether element e exists in array a.

```
./runTh -t eina.th
```

You can always learn about the different options in the tool with the following command.

```
./runTh -h
```

Now answer the queries depending on whether you think the element e is in the array presented to you between $left$ and $right$. Note that the variable `a.size_minus_1` expresses the value `a.size() - 1`; in other words the last valid index in the array. *Hint: when left and right are outside bound, you need to answer with 'N'.*

When you answer with 'N' you get choices to tell why you think the presented assignment is bad. This is an optimization that helps speed the process. If you are in doubt press 'C' to ignore the optimization options.

Do not use these optimizations in the first session.

The 'B' optimization presents you with clauses and their values that led to the assignment presented to you. You can pick a subset of these clauses and their assignments that make you decide that the assignment is false.

The 'V' optimization allows you to present the variable names whose combination of values you think are bad. Do not use this option if one of those variables was an array, that is not supported yet.

After your done with the experiment, look at the file `eina.th` with the vim or your preferred editor. (you can edit on windows and copy and paste into the terminal.)

The file looks like the following.

```
theory ls {

  //declaration section
  int [] a;
  int left;
  int right;
  int e;
  local int i;

  constants {0, 1, -1}

  grammar {
    (a,left,bound);
    (a,right,bound);
    (e,a,=);
    (a,i,index);
  }

  // user defined vocab section
  vocab {
    (((0 <= left) and (left <= right)) and (right <=a.size_minus_1));
    ((left <= i) and (i<= right));
    (a[i] = e);
  }
}
```

That file told the tool the type theory through the declaration, it knew it needs a local variable to specify the predicate and declared as a `local` variable, and also specified a vocab.

We will now try the same experiment but with a limited theory and let the tool decide for us. told it the vocab. Si

Now you will try the same approach with simply the types

1 Injecting the quantifier

Now look the theory file `eina.free.th`. It only has the types of the variables that are input to the desired predicate. We will use that file with the quantifier injection ability of the tool to get the same predicate.

Run the same way now but with the following command.

```
./runTh -t eina.free.th -b -e
```

2 Build a linear search post condition

The theory file `lse.th` has the type theory of an array a, a $left$ bound, a $right$ bound, an element e, and a return value rv. It also uses the result from the `eina` experiment as a Boolean predicate.

Run the experiment and answer whether the values presented to you satisfy the following.

The value of rv is inclusively between the valid indices $left$ and $right$ when the value of element e exists in a. The value if rv is -1 when the element e is not in the array between $left$ and $right$.

The command is to run is:

```
./runTh -t lse.th
```

All you need is to plug in the specification for `eina` from the previous experiment to get a universal quantifier.

3 Build the post condition without providing the vocab

Now you will write the specification without providing the tool with the vocab. We will use the file `lse0.th` for that.

```
./runTh -t lse0.th -b
```

4 Build the specification from scratch

The tool allows you to build the whole linear search postcondition from scratch by building a quantifier free formula equivalent to it. You only need to provide it with the type theory. We will leave this till the end.

To do that check the file `ls.th` and run with the following command.

```
./runTh -t ls.th -b -q
```

5 More specifications

Pick one or more of the following and write specifications for them using the tool. You can chose whether to start from your own vocab or just use the type theory to write the specification.

5.1 Array in order property

Write a theory file `inorder.th` to specify that valid elements in array a are in order.

Hint: you need to declare the array, and you need to add a rule that the elements of the array could be compared to each other in the grammar. $(a, a, <=$ $)$

Run your theory file when you are done. Make sure to pass the "-b" to have the tool build the vocab.