ETH zürich

Alexander J. Summers

Program Verification

Exercise Sheet 5: Encoding to SMT

Assignment 1 (Sequence Take and Drop)

Define appropriate axioms for defining `length` and `lookup` properties with respect to `take(s,n)` and `drop(s,n)` functions (representing the standard operations which take the first `n` elements or drop the first `n` elements of a sequence, respectively). For example, it should be possible to prove (sequence elements are indexed from 0):

```
assume length(s) >= 5;
lookup(take(s,3),2) == lookup(drop(s,2),0)
```

Note that you should choose appropriate triggers for your defined axioms.

Is there a potential for matching loops in the axioms you've defined? Can you think of test cases in which they would be incomplete (i.e. it would be impossible to prove a property which should be true)?

Assignment 2 (Extensionality)

Extensional equality is the idea that if two mathematical objects are *observationally equal*, they should be known to be equal objects. Another way of looking at this is that there is not redundancy in the mathematical type in question.

For the sequence type discussed in the lecture, the two key "properties" are the length of a sequence and its behaviour when we apply `lookup` to it. Write down an axiom which expresses extensional equality for sequences (if you get stuck, one is shown in the lecture slides).

One complete (but expensive) way to instantiate the extensional equality axiom, is to do so for *every pair* of sequences encountered in our particular problem. Is it in general sufficient to compute these instantiations by simply taking the set of ground, sequence-typed terms in the input problem, and adding these instantiations eagerly?

Suppose instead that we add a function `isSequence(s:Sequence):Bool` along with an axiom defining this function to be true for all input values (write down this axiom). Now, for each atom in the original input problem (i.e. a formula containing no propositional connectives), and for each sequence-typed term `t` in the ground atom, conjoin the formula `isSequence(t)`. For example, the formula:

```
forall s1:Sequence, s2:Sequence ::
  length(append(s1,s2)) == length(s1) + length(s2)
```

would become

```
forall s1:Sequence, s2:Sequence ::
  length(append(s1,s2)) == length(s1) + length(s2) &&
  isSequence(append(s1,s2)) && isSequence(s1) && isSequence(s2)
```

What triggers could you then choose for your extensionality axiom? How many instantiations of this axiom would result?

Assignment 3 (Axiomatising Maps)

Write an axiomatisation for (total) mathematical maps from integers to integers. Your encoded type should include a representation for two operations: *map lookup* (sometimes written $M[i]$, which looks up a particular integer key in the map M, returning the mapped-to integer), and *map update* (sometimes written $M[u \mapsto v]$), which is a map with the same lookup behaviour as M, except for the key u which is mapped to v.

Can you extend your axiomatisation to support a "bulk update" on maps, such that all keys satisfying a certain condition are updated to a certain value?