

Assignment 4 - Alloy

Exercise 1

Consider the following Alloy model of a counter:

```
module util/integer

sig Counter {
  n: Int
}

pred inc[c, c': Counter] {
  c'.n = c.n.add[Int[1]]
}
```

The predicate `inc` models the increment operation of the counter.

1. Use the abstract machine pattern that you have seen in the lecture to define the traces of the counter; see slides 138–144. For this task you must use the Alloy library `util/ordering` as shown on the slides. Initially, the value of n must be 0. Define the initial state of the counter using the predicate `init[c: Counter]`.

Note that Alloy's integers are signed, have limited size and can overflow. You can specify the size of the integers explicitly; e.g., `run show{} for 10 but 5 Int` will generate instances of the predicate `show` with at most ten instances of every signature and 5-bit signed integers.

2. Express the following two invariants using assertions:

inv1: The counter's value is always greater than or equal to zero.

inv2: The counter's value never decreases.

The first invariant **inv1** is a state invariant. You must therefore check whether it holds for all states. The second invariant **inv2** is a temporal invariant. You must check whether for any two states c and c' where c is a predecessor of c' , $c.n \leq c'.n$ holds. You can use the predicate `lt[c, c']` to check whether c is a predecessor of c' .

Exercise 2

Recall the two `ImageFile` class implementations from the lecture. You are given the Alloy model for the `ImageFile` implementation with eager initialization; see the file `imagefile_eager.als`. Consider the following two invariants:

inv1: `getImage()` always returns a non-null value.

inv2: `getImage()` always returns the same value.

You have two tasks:

1. Define the initial state and the traces for the `ImageFile` model. Model the two invariants as assertions and check whether they hold.
2. Create an Alloy model for the alternative `ImageFile` implementation which uses lazy initialization:

```
class ImageFile {
  String file;
  Image image;
  ImageFile(String f) {
    file = f;
  }
  Image getImage() {
    if(image == null) {
      // load the image
    }
    return image;
  }
}
```

Observe several traces to ensure that your model has the intended behavior. Check whether the invariants that you defined in (1) hold for your model.

Hint: In the eager version, the method `getImage()` had no side effects (did not change the program state) and could therefore be modelled as a function with a `State` parameter. In the lazy version, calling `getImage()` changes the program state, and must therefore be represented as a predicate that relates the previous state to the next one.

Exercise 3

Consider the following Alloy model:

```
sig Node {
  next: Node
}
assert demo {
  all n: Node | some m: Node | m.next = n
}
```

For the given model we have one constraint: each node must have exactly one next node.

The assertion `demo` holds iff it is satisfied in all instances that satisfy the model's constraints. To check whether the assertion holds, Alloy searches for a counter-example (i.e., an instance where the model's constraint is satisfied and the assertion `demo` is violated).

1. Encode the constraint and the assertion that correspond to checking `check demo for 1` into a boolean formula.

Hint: Given two nodes n and m , introduce a boolean variable $x_{n,m}$ to denote $(n, m) \in next$. Treat the universal and existential quantifiers as conjunction and disjunction, respectively.

Check if the boolean formula has a satisfying assignment. If it is satisfiable, give a counter-example where the assertion is violated.

2. Repeat step (1) for `check demo for 2`.
3. Add a field `prev: Node` to the signature of `Node`, and also the fact `all n: Node | n.next.prev = n` to the Alloy model.

Repeat steps (1) and (2).

Does there exist a scope, possibly larger than 2, in which the assertion is violated?