Assignment 2
Count On It: Unary Verification

15-414: Bug Catching: Automated Program Verification

Due 23:59pm, Friday, February 9, 2024
65 pts

This assignment is due on the above date and it must be submitted electronically on Gradescope. Please carefully read the policies on collaboration and credit on the course web pages at `http://www.cs.cmu.edu/~15414/assignments.html`.

What To Hand In

You should hand in the following files on Gradescope:

- Submit the file `asst2.zip` to Assignment 2 (Code). You can generate this file by running `make handin`. This will include your solution `unary.mlw` and the proof session in `unary/`.
- Submit a PDF containing your answers to the written questions to Assignment 2 (Written). You may use the file `asst2-sol.tex` as a template and submit `asst2-sol.pdf`. You can generate this file by running `make sol` (assuming you have pdflatex in your system).

  Make sure your session directories and your PDF solution files are up to date before you create the handin file.

Using LaTeX

We prefer the answer to your written questions to be typeset in LaTeX, but as long as you hand in a readable PDF with your solutions it is not a requirement. We package the assignment source `asst2.tex` and a solution template `asst2-sol.tex` in the handout to get you started on this.

1 The Count's Number of the Day (50 pts)

Unary counters, based on the unary numeral system, use a single symbol to represent numbers in a linear, additive fashion. In this problem, you are asked to implement and verify some simple functions over unary counters.

> **Each function you write should be verified against contracts expressing the correctness of your implementation.**

Consider the following type `ucount` (which you can find in the file unary.mlw) that represents unary counters:

```
type ucount = Z | S ucount
```

`Z` represents zero, and it is the base case for the unary counter, analogous to the digit 0 in the decimal system. `S ucount` represents the successor, indicating the next number of the sequence. For example, if you have a counter representing the number two, applying `S` to it would produce a representation of the number tree. In this system, natural numbers are constructed by repeatedly applying the successor constructor `S` to the base `Z`. For instance, the number three would be represented as `S (S (S Z))`, indicating three applications of the successor function to zero.

To reason about unary counters, we will use a data structure `ctr` with a ghost field `model` that will help us verify some simple functions over unary counters. More information about data structure and ghost variables is available in the lecture notes from Lecture 3 and Lecture 4.

```
type ctr = {counter: ucount ; ghost model: int}
```

In `ctr`, the ghost model corresponds to the integer value of the unary counter.

Task 1 (10 pts). Specify a predicate `eq_ctr_int` that relates a unary number to its integer value by a set of axioms. **Hint:** you may want to specify one axiom for each case, i.e., zero and successor.

```
predicate eq_ctr_int (c:ucount) (n:int)
```

Task 2 (5 pts). Write data structure invariants that relate the unary counter `counter` with the ghost `model`. Do not forget the `by` clause of the data structure invariant.

Note: Your contracts for the following functions **must** be related to the ghost model. You can also use the functions that you already defined when writing the code for other functions.

Task 3 (5 pts). Define a function `is_zero` that returns true if counter `c` is zero and false otherwise.

```
let is_zero (c: ctr) : bool
```

Task 4 (10 pts). Define a function `succ` that returns the successor of the counter `c`.

```
let succ(c:ctr) : ctr
```

Task 5 (10 pts). Define a function `compare_ctr` that compares two unary counters and returns true if they represent the same number or false otherwise.

```
let compare_ctr (c1: ctr) (c2: ctr) : bool
```

Task 6 (5 pts). Define a function `to_int` converting a unary number counter `c` to the integer it represents.

```
let to_int (c:ctr) : int
```

Task 7 (5 pts). Define a function `to_ctr` converting an integer a to a unary counter.

```
let to_ctr (a:int) : ctr
```

2 It's a Question of Semantics (15 pts)

In this collection of problems we work with the simple while language from Lecture 5.

Task 8 (10 pts). Conjecture the semantics of the following program (let's call it α_0):

```
?(n >= 0) ;
x <- n ;
y <- 1 ;
while (x > y)
( x <- div (x+y) 2 ;
  y <- div n x )
```

where div is integer division. You should describe your conjectured semantics in terms of the relation between ω and ν in

$$\omega[\![\alpha_0]\!]\nu$$

For the while loop, describe ω_{init} at the beginning of the loop and ω_{done} at the end, but you do not need to describe the intermediate states.

Task 9 (5 pts). Define the semantics of a for-loop

$$\mathsf{for}\ x\ e_1\ e_2\ \alpha$$

which goes through the values for x between the values of e_1 and e_2. It starts at the value of e_1 and counts up or down to the value of e_2, inclusively, executing α each time.