Sagar Kumar
Bremen, 2017

Secure and Dependable System Homework 5

Question 1:

A class invariant in a condition that always holds true before and after the execution of any method in the class. There are three methods available in the class. Namely, getSize, push and pop. Let's look at all of them individually. But first let's make sure the class invariant holds even before any method is called.

We start with the initial conditions for an initialized stack. IE no elements (empty stack) and size 0. Obviously, the class invariant holds here and empty stack implies length(elements) as 0, also the initial value of size.

Now let's call a method…

getSize: This one is easy in that no operations are done on the stack itself. The function merely returns the size of the stack. Therefore length(element) will always equal size given that the condition held true before calling the function.

Push: Two statements are executed in this method.

$$elements := cons(x, elements)$$

$$size = size + 1$$

As we execute this function, we see an element (namely, x) entering the stack and immediately the stack size is incremented. Hence

$$length(elements) + 1 = size + 1$$

Pop: In this function, if the stack is empty (ie initial conditions), the class invariant holds. We have already proven this to be true. However if the stack is not empty we execute the following:

$$e := elements.head$$

$$elements = elements.tail$$

$$size := size - 1$$

$$Some(e)$$

Note that after we pop, we immediately decrement the size of the stack such that $length(elements) - 1 == size - 1$.

However lets take a look at the class as an entirety and execute more than one method and observe the behaviour of the class invariant. Suppose we push and element (say 'x') and pop it.

We have:

$$Let: length\left(pop\left(push(x)\right)\right) = size$$

$$and\ push = size + 1$$

$$and\ pop = size - 1$$

$$Then, pop\left(push(x)\right) = \left((size + 1) - 1\right)$$

$$pop(push(x)) = size + 1 - 1$$

$$pop(push(x)) = size + 0 = size$$

Therefore: $length(size) == size$

Therefore we have confirmed that our class invariant holds even when we execute more than one method.

Sagar Kumar
Bremen, 2017

Question 2:

Weak class invariant: (ie for general case):

$$(month \geq 1 \ \wedge \ month \leq 12) \ \wedge \ (day \geq 1 \wedge day \leq 30) \wedge (year \geq 0)$$

Code on github repo… “date.cpp”

Sagar Kumar
Bremen, 2017

Question 3:

Induction rule…

<table>
<tr><td>By “zero_right” rule</td><td>By “zero_left” rule</td><td></td></tr>
<tr><td>$\vdash m + zero \rightsquigarrow m$</td><td>$\vdash zero + m \rightsquigarrow m$</td><td>Proof given below…</td></tr>
<tr><td colspan="2">$m: nat \vdash m + zero == zero + m$</td><td>$m: nat, n: nat\ p: \textbf{proof}\ m + n == n + m \vdash succ(m) + n == n + succ(m)$</td></tr>
<tr><td colspan="3">$m: nat, n: nat \vdash m + n == n + m$</td></tr>
</table>

Proof p:

Induction case:

Goal to prove:

$$succ(m) + n == n + succ(m)$$

Proof:

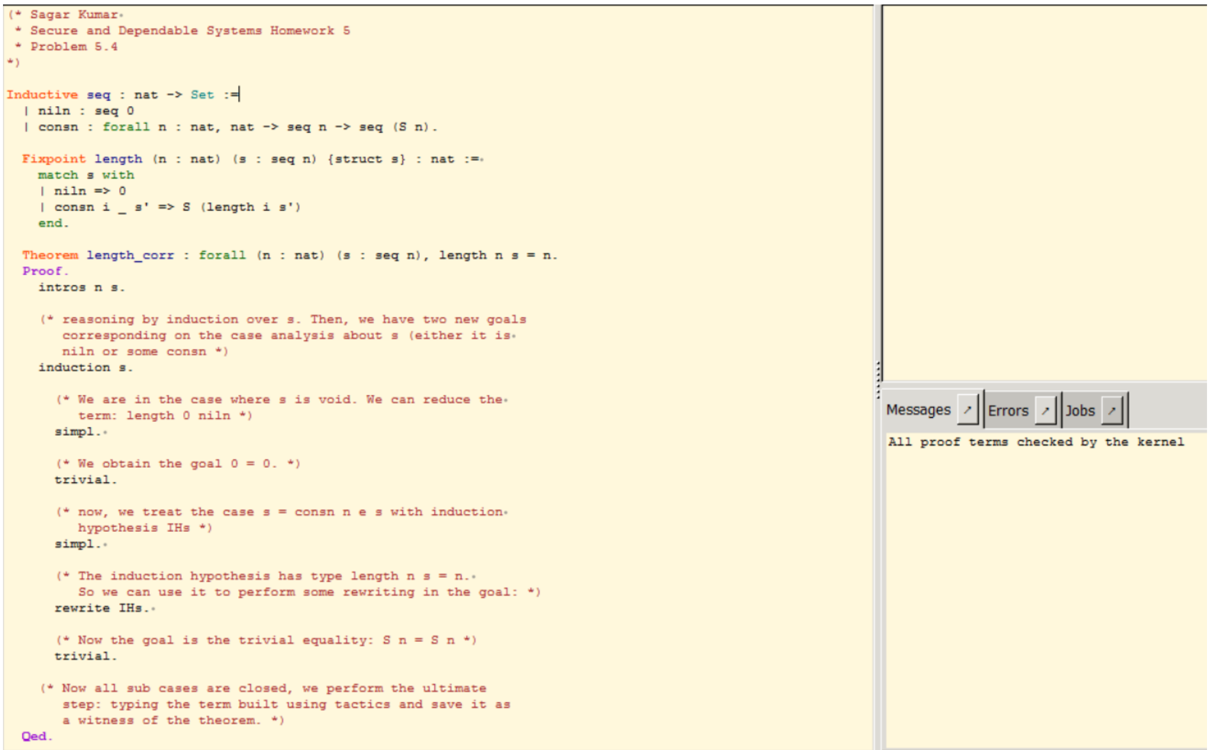
1. $succ(m) + n \rightsquigarrow succ(m + n)$ (by succ_left)
2. $succ(m + n) \rightsquigarrow succ(n + m)$ (by Induction hypothesis)
3. $succ(n + m) \rightsquigarrow n + succ(m)$ (by succ_right)

   Therefore, $n + succ(m) == succ(m) + n$ and vice versa.

Sagar Kumar
Bremen, 2017

Question 4:

I have attached the screen shot below. As you can see I have made my own program (at the top of source code is my name and class name to prove this is mine) and I compiled it. The program returned the message “All proof terms checked by the kernel”.



```
(* Sagar Kumar
 * Secure and Dependable Systems Homework 5
 * Problem 5.4
*)

Inductive seq : nat -> Set :=
  | niln : seq 0
  | consn : forall n : nat, nat -> seq n -> seq (S n).

  Fixpoint length (n : nat) (s : seq n) {struct s} : nat :=
    match s with
    | niln => 0
    | consn i _ s' => S (length i s')
    end.

  Theorem length_corr : forall (n : nat) (s : seq n), length n s = n.
  Proof.
    intros n s.

   (* reasoning by induction over s. Then, we have two new goals
      corresponding on the case analysis about s (either it is
      niln or some consn *)
    induction s.

      (* We are in the case where s is void. We can reduce the
         term: length 0 niln *)
      simpl.

      (* We obtain the goal 0 = 0. *)
      trivial.

      (* now, we treat the case s = consn n e s with induction
         hypothesis IHs *)
      simpl.

      (* The induction hypothesis has type length n s = n.
         So we can use it to perform some rewriting in the goal: *)
      rewrite IHs.

      (* Now the goal is the trivial equality: S n = S n *)
      trivial.

   (* Now all sub cases are closed, we perform the ultimate
      step: typing the term built using tactics and save it as
      a witness of the theorem. *)
  Qed.
```