Notes on writing proofs and reasoning about code
15-122 F2011

Jamie Morgenstern

September 5, 2011

Writing proofs is an art. Mathematics has any number of proofs which are beautiful, and others which are clever and subtle. The former should show you that there are ways to write proofs using good taste; the latter that there is no single formulation which is general-purpose enough to prove all mathematical statements, and that many proofs require a certain amount of inspiration and luck. Fortunately for us, the proofs we'll be doing in this class will tend to follow a particular pattern, which will (usually) allow us to work within the following framework.

1 Outline for constructing proofs

Writing these proofs will be an iterative process; in particular, 1,2 and 3 will likely need several back-and-forths for difficult proofs (if you can't prove a theorem, it might be that your assumptions aren't strong enough, or that you need more loop invariants. If you can't prove your loop invariants, you might have picked them to be stronger than you need for the proof, such that they aren't even true).

- Assumptions
  What are our assumptions at the onset of our proof? If we're trying to show a statement "If A, then B" is true, "A" is an assumption, and using A, we try to show B must hold. These take the form of '//@require' statements in C_0.
- Write down Loop invariants
  What statements should be true after each iteration of a for or while loop? What are the relationships between the variables? Which pointers are non-null, which integers are positive, etc? These take the form of '//@loop_invariant' statements in C_0.
- Prove Loop invariants
  First, show that the loop invariants you wrote down in the previous step start before the loop is executed. Then, assume the invariant holds when you enter the loop. Show that, at the end of the loop body, the invariant still is true.
  For those of you familiar with induction, this should feel familiar; this is a bit like proving the base case, and then using an induction hypothesis to show it holds for all number of iterations of the loop body thereafter.
- Use the loop invariants, assumptions, and loop exit requirements to show your conclusion
  Now that you've proven your loop invariants correct, showing the return value of your function (using '//@ensures' statements in C_0), consider your assumptions, your loop invariants, and the conditions that caused you to break out of any and all loops in the function. If you stopped executing a loop, this means either (a) the function returned from within the loop, or that the guard failed to pass (so you know the guard is false, and you can reason from that).