1 Sample Proof

In this course, when we prove the correctness of a program we use a combination of the logic reasoning rules we learned when studying propositional and first-order logic and the symbolic execution rules we learned studying the formal semantics of OCaml programs. When we have a proposition that we wish to prove of an OCaml program, we will generally:

1. Use our rules of natural deduction to break down the proposition into an atomic proposition consisting of an equality proposition (=) between OCaml programs.
2. Use symbolic execution to evaluate the OCaml programs and show that the equality proposition holds.

While we could build a bottom-up formal proof tree in the style of natural deduction, our program correctness proofs will be far more complex than the propositional logic proofs we have explored previously. As a result, we will write our proofs using a combination of prose and symbols in a more natural top-down manner.

To demonstrate this style, consider the proposition we discussed in class regarding the employee algebraic datatype and its corresponding pay function:

Claim 1. \( \forall e, n, a1, a2, s1, s2. e \neq Manager("Tony", a1, s1) \rightarrow pay (CEO(n, a2, s2)) \leq pay e \)

Proof. We prove this claim by symbolic evaluation of the OCaml functions contained in the claim, using the proof recipe described above.

- To prove the claim \( \forall e, n, a1, a2, s1, s2 . . . \), it suffices to assume that we have variables \( e, n, a1, a2, s1, s2 \) of the appropriate types and go on to prove the implication following the quantification.
- To prove that \( Manager("Tony", a1, s1) \rightarrow pay (CEO(n, a2, s2)) \leq pay e \), we assume the premise of the implication (call it \( pre \)) and go on to prove its conclusion.
- To prove that \( pay (CEO(n, a2, s2)) \leq pay e \), we symbolically evaluate both sides of the inequality.

\[ \text{pay (CEO(n, a2, s2)) evaluates as follows:} \]

\[ \text{let account_base = 1 in . . .} \]

\[ \text{let prog_base = 5 in . . .} \]
---> match (CEO(n, a2, s2)) with ...  
---> 1

So the left-hand side of the inequality is 1.

- pay e evaluates as follows:

  pay e
  --> let account_base = 1 in ...
  --> let prog_base = 5 in ...
  --> match e with ...

To proceed, we must perform case analysis on possible shapes of e:

* e = Account(n1, a3, s3). Then match e with ... evaluates to if s3 == "none"  
  1 + 2 else 1. To evaluate further, we perform case analysis on the guard of the conditional. If s3 == "none" is true then pay e →* 3 and we have 1 ≤ 3 which is true. If s3 == "none" is false then pay e →* 1 and we have 1 ≤ 1 which is true.

* e = Programmer(n1, a3, s3). Then match e with ... evaluates to if y == 0  
  then ... To evaluate further, we perform repeated case analysis on each of the guards of the conditionals. In the first case, y == 0 is true so pay e →* 5 and we have 1 ≤ 5 which is true. In the second case, y == 0 is false and not (is_senior e) so pay e →* 10 and we have 1 ≤ 10 which is true. In the final case, both y == 0 and not (is_senior e) are false so pay e →* 20 and we have 1 ≤ 20 which is true.

* e = Manager(n1, a3, s3). Then match e with ... evaluates to if n = "Tony"  
  then ... Because of our assumption pre we know that n = "Tony" is false so we evaluate to then branch of the conditional. To evaluate further, we perform case analysis on the guard of the second conditional. If not (is_senior e) is true, then pay e →* 20 and we have 1 ≤ 20 which is true. If not (is_senior e) is false, then pay e →* 40 and we have 1 ≤ 40 which is true.

* e = CEO(n1, a3, s3). Then match e with ... evaluates to 1 and we have 1 ≤ 1  
  which is true.

In all cases, we have shown that the desired inequality holds, therefore the over claim holds.

□

This may feel like an intimidating piece of writing, but don’t be deterred! The writing is simply formalizing our intuition as to why the claim holds—we know the inequality holds by case analysis on e and evaluation of the resulting program, noting that the precondition of our function precludes the one case where the inequality would not hold. The length of the formal proof merely comes from following through precisely why our intuition holds using the formal rules we’ve built up in the course so far.
2 Words of Advice

Now that we are writing formal proofs of considerable size, it may be intimidating to determine how to proceed. Here is some advice to help you out on this journey:

- Recall that a proof is composed of a goal and a set of assumption you have acquired throughout the proof process. Be diligent about tracking these two things throughout your proof—at all times, you should know what your proof goal is and what your assumptions are. If you are unsure how to proceed, you should back up your proof to the point where you know these two things and then try to make progress again from that point.

- Don’t try to \LaTeX your proof immediately. Use paper, write down your thoughts, make notes, erase or scratch things out, and only once you think you have a complete proof should you transfer it to \LaTeX. The act of transcribing your proof to \LaTeX will serve as a final check that your logic is sound.

- Mathematical proof is (partially) algorithmic, like programming! The reason that we’ve spent so much time discussing natural deduction and symbolic execution is so that you have something to do at each step of a proof. If you are ever stuck, think about what reasoning principle or rules apply to your current goal. If a logical connective is involved, how do you reason about that logical connective? If a program is involved, how do you evaluate that program?

- When reasoning symbolically about programs, you will arrive at points where evaluation gets stuck, for example, you don’t know the concrete value of the guard of a conditional or pattern match. To make progress in these cases, you need to perform case analysis on the guard as shown in the above proof.

- A final note, mechanical proofs of this size are really tests of your mathematical fortitude. Can you carry rigorous, logical thinking from the start of a proof to its end? Keep in mind that this is the truly valuable meta-skill that you are refining as you work through these proofs. Force yourself to be as rigorous and logical as your author your proofs.