Damian Franco

dfranco24@unm.edu

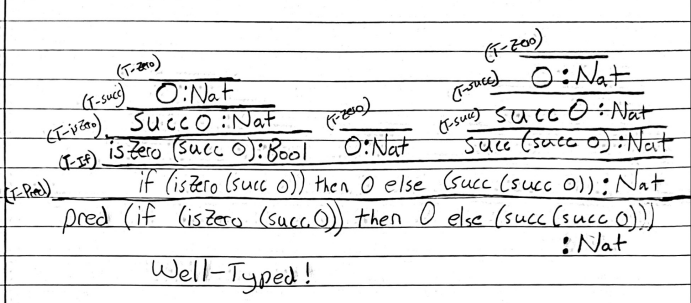
101789677

CS-558 001

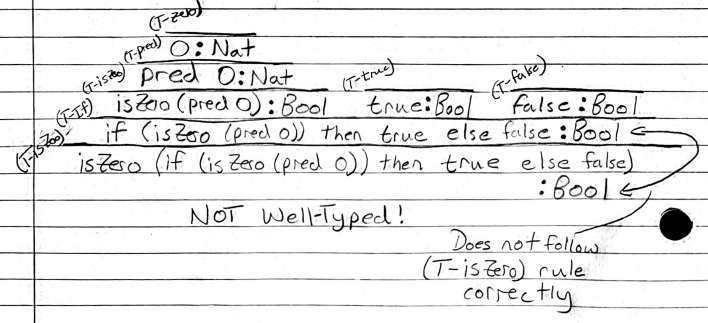
Homework 4

4.1 Typing in the typed calculus of numbers and booleans

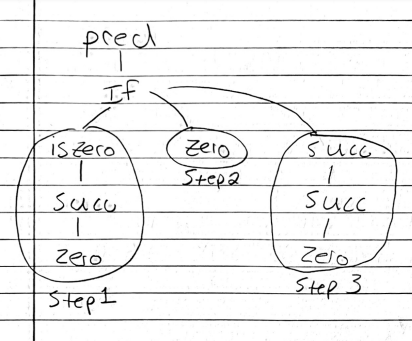
1. For the first question, we were asked to use the typing rules to determine whether each of the following terms is well-typed. We were given the term *pred(isZero(succ 0)) then 0 else (succ(succ 0)))*. Right away, I made the assumption that this term will be well-typed and it was found that the term ***is well-typed***. I was able to come to that conclusion because of first applying the *T-pred* rule and extracting the if statement. After, I used the *T-if* rule to extract further the three terms within the if statement and found that each of them followed the proper *T-if* rules where the first term *t1* was of type *Bool* and the last two terms, *t2* and *t3* had the same type. In this instance it was type *Nat*. The rule *T-pred* also expects a type *Nat* which the if statement fulfills this type. Below you can see my work in type checking.

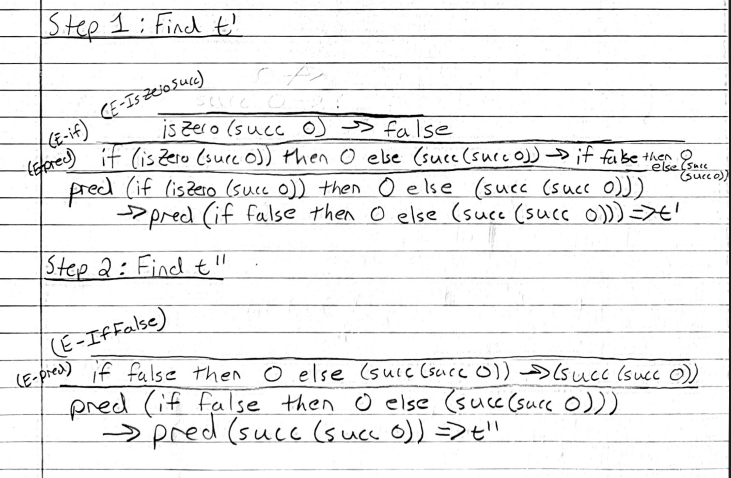


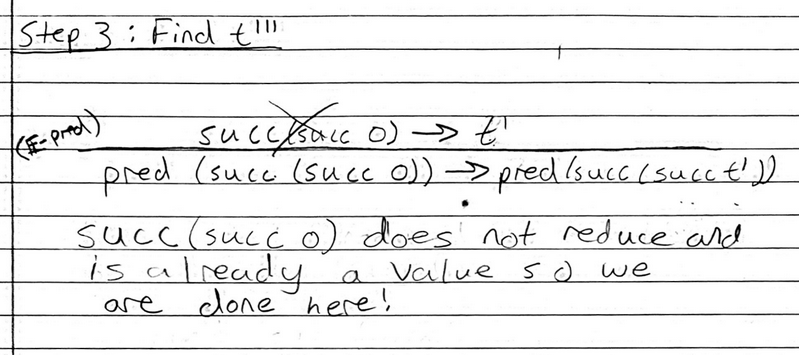
1. For the next question we were given another term to type check on. The term we were given was *isZero(if(isZero(pred 0)) then true else false)*. I first assumed that this term would not be well-typed and found that it was indeed ***NOT well-typed***. I still extracted each term far enough to find exactly why it was not well-typed by first extracting the if statement with the *T-isZero* rule. Next I checked each type of the terms within the if statement with the *T-if* rule and found that all three terms were *Bool*. this is not a problem, but is the source of the problem because with terms *t2* and *t3* being of type *Bool* and the term *isZero* expecting the a 0 or a numeric value (*nv*), then it does not correctly fulfill the right type rules with the rule *T-isZero.* Below you will see my work and how I found that this term was not well-typed.



4.2 Evaluation in the typed calculus of numbers and booleans

1. For the first question, we were asked to use the evaluation rules for the terms given to us to construct a full derivation for each step of evaluation and identify when the term has either become stuck or been reduced to a value. The first term that was given to us to derive is *pred(isZero(succ 0)) then 0 else (succ(succ 0)))* which is a well-typed term that we found in the first question. First, I developed a “tree” of the terms that were needed to derive. This allowed me to get a better understanding of what to do. For step 1, the first derivation *t’* must be found through using the evaluation rules. The first rule applied was the *E-pred* which does not have an exact derivation term right now because we need to reach an axiom to form one. The *E-if* rule was applied to check for the derivation of the term *t1* which happens to be *isZero(succ 0)*, which also happens to be an axiom with the *E-isZeroSucc* rule.The axiom reduces down to *false*, this would then trickle down the “tree” here which would lead to the finding of *t’*. After finding *t’*, the next step was to find the next derivation *t’’* which I used the same as the *E-ifFalse* rule to derive the third term from the if statement. This third term looks like a set up for returning a value, which I found out it was when I tried to derive *t’’’*. This was because *succ(succ* *0*) does not reduce anything, rather it is a value itself, just like the term *succ 0* which also does not reduce but it is already a value as well which is why in this term evaluation is reduced to a ***VALUE*** at the third term evaluation.





1. For the first question, we were asked to perform reductions on the not well typed term *isZero(if(isZero(pred 0)) then true else false)*. I also created a “tree” like structure here to make the reductions similar to myself. I then started with step 1 or finding *t’*, this consisted of using the *E-isZero* rule and the *E-if* rule to extract the inner terms and check if they can be reduced. The inner terms in this case will reduce to the term *0* from the *E-predZero* rule and *true* with the *E-isZeroZero* rule. This will then trickle down to reduce to a new term that is *true* in the first sub term of the if statement. The next step to find *t’’* was then done with two rules. The first was the *E-ifZero* rule being applied to extract the if subterm. The next rule applied was the *E-ifTrue* rule that reduces the if statement down to the then term of the if statement. This term was also a *Bool* and was *true*. That means that *t’’* was found which is simply *isZero(true)*. This is where the problems also occur because the next reduction cannot occur. The term *isZero(true)* cannot be reduced because the term has no rule to reduce *t’’* down further which results in this reduction being ***STUCK***. Below, and the next page, you will see my work for this problem.

