

FPP Midterm Report

Yale-NUS College AY 17/18

Jeremy Yew
A0156262H

7 October 2017

1. Table of Contents

1.	Table of Contents	2
2.	Introduction	3
3.	Left-balanced binary trees	3
3.1	Proposition: projecting_an_embedded_binary_tree	3
3.2	Proposition: projectable	3
4.	Product of three consecutive numbers	4
5.	Mystery functions	4
5.1	there_is_at_most_one_mystery_function_k	4
5.2	there_is_at_least_one_mystery_function_k	5
5.3	there_is_at_most_one_mystery_function_l	5
5.4	there_is_at_least_one_mystery_function_l	5
5.5	there_is_at_most_one_mystery_function_m	5
5.6	there_is_at_least_one_mystery_function_m	5
6.	Conclusion	5
7.	Acknowledgements	6

2. Introduction

In this report I explain the approaches taken and learning points gained from the process of solving each assignment part. Assignment specifications can be found at: http://users-cs.au.dk/danvy/YSC3236.2017-2018_Sem1/lecture-notes/week-07_midterm-project.html.

3. Left-balanced binary trees

3.1 Proposition: projecting_an_embedded_binary_tree

I was initially stuck as the specific unfold lemma `unfold_project_Node_Tree_Leaf` I was trying to write was too strong: it assumed that `project_binary_tree_into_left_binary_tree t1` returned `Some lt1` instead of `None` which is unprovable without any additional propositions. I tried to use the fact that `t1` is left balanced, but then realized that this was identical to the second proposition `projectable`. The solution was simply to write `unfold_project_Node_Tree_Leaf` with the entire match expression; the induction hypothesis provided the fact that projecting an embedded `lt` returns `Some lt`, and this could be directly applied to the match expression in the goal.

3.2 Proposition: projectable

Explanation for my original, indirect proof:

The main challenge was trying to write a lemma about `project_binary_tree_into_left_binary_tree (Node t1 n Leaf)`. As it turned out the lemma `project_t1_with_existential_lt1` was kind of like a paraphrase of the original function, and was easy to prove after remembering the tactic `destruct` which helped me get rid of the existential in the assumption (this helped in other proofs I had been stuck in, such as in Part 2).

The tricky part was realizing that I had to define two distinct existentials in the lemma (and separate their scope via parentheses, otherwise the first existential applies to the second proposition as well). Previous iterations (see `project_t1_with_existential_lt1_initial`) involved the single existential `lt1` shared across both propositions, which seemed to make sense because in the definition, we have the same `lt1` in both the match and the result. However as I realized eventually I was not able to apply the lemma in the inductive hypothesis (`Error: Statement without assumptions`) because the inductive hypothesis contained a single existential over its entire proposition, whereas the single existential in the lemma was applied over both propositions, which made it a single statement with no separate assumptions. The separate existentials in my final lemma ensured the inductive hypothesis matched the first proposition in the lemma exactly.

Explanation for canonical, more direct proof (after `Restart`):

The canonical proof involves no other lemmas besides the basic unfold lemmas. Instead of proving that `t2` must be a `Leaf`, we consider the cases where `t2` is a `Leaf` or `Node`; when it is a `Node` we know that `Node t1 n t2` cannot be left balanced, and we use discriminate on the false proposition. In the case that it is a `Leaf`, instead of writing a lemma about `project_binary_tree_into_left_binary_tree` (`Node t1 n Leaf`) that the inductive hypothesis may be applied to, we rewrite the goal into a match expression involving just `t1` using `unfold_project_Node_Tree_Leaf`, and then directly apply the inductive hypothesis on `t1` (given the assumption that `t1` is left-balanced, and after separating the existential from `H_t1` via `destruct`) on the match expression in the goal. This is actually similar to the approach taken in Proposition: `projecting_an_embedded_binary_tree`. The reason why I did not complete this simpler solution initially was because, although I had attained the assumption I wanted to apply directly to the match expression, I had not remembered about `destruct` at that point, and thus could not apply the assumption to the match expression as it contained an existential.

4. Product of three consecutive numbers

I proved this mathematically rather than computationally. Writing a proof out on paper helped. The `destruct` tactic was crucial in allowing me to apply the inductive hypothesis. The lemma `SSSn_is_3_plus_n` made the main proof more elegant as it targets only the argument that I need to rewrite; otherwise I would have rewritten the other two arguments as well into arabic form, which would mean having to rewrite them back into Coq's successor form before applying the inductive hypothesis.

This proof is similar to the proof of the proposition that the product of two consecutive numbers is even, involving the same approach in the inductive case: Expand using the last term ($n + k$) to get an addition of two parts, the first part matching the terms in the inductive hypothesis and the second part being a multiple of k . Then factorize both by k .

5. Mystery functions

5.1 `there_is_at_most_one_mystery_function_k`

Since the Fibonacci sequence is a recurrence relation involving the previous two terms (rather than just the previous term), we would need to define an induction principle which uses two inductive hypotheses instead of just one. In this case, since the recurrence relation in `specification_of_the_mystery_function_k` contains `mf` (`S (p`

$+ q)$, which can be applied to $f(S(S n))$, we know that the inductive step should be to $P(S(S n))$. In addition the LHS of the recurrence relation contains $mf(S p)$ and $mf(p)$, hence the inductive hypotheses should be $P(n)$ and $P(S n)$.

5.2 there_is_at_least_one_mystery_function_k

Arithmetic manipulation.

5.3 there_is_at_most_one_mystery_function_1

We need to define a different inductive hypothesis in this case; since the recursive relation in `specification_of_the_mystery_function_1` contains $mf(S(S(S n'')))$, we know that the inductive step should be to $P(S(S(S n)))$. In addition the other two terms of the recurrence relation are $mf(n'')$ and $mf(S(S n''))$, hence the inductive hypotheses should be $P(n)$ and $P(S(S n))$.

5.4 there_is_at_least_one_mystery_function_1

Have not attempted.

5.5 there_is_at_most_one_mystery_function_m

Proving this via induction is impossible as the specification is not inductive, and we cannot use an inductive hypothesis. We could, however, define another specification that *is* inductive, prove that there is at most one mystery function that satisfies that specification, and then prove that it is equivalent to `specification_of_the_mystery_function_m`.

5.6 there_is_at_least_one_mystery_function_m

The `mystery_function_m` and `mystery_function_m_aux` flattens a tree to the right via the use of an accumulator. The accumulator enables the recursive function to achieve a transformation that involves successive ‘re-shuffling’ or re-constructing of the object to be returned (instead of inductively constructing at return time). The proof was solved via basic unfold lemmas.

6. Conclusions

- Sometimes the solution is quite simple. So always make sure to write the basic unfold lemmas and do the canonical approach first.
- Pen and paper, as always.
- Name everything.

- To achieve optimal progress I need to spend a bit of time every day on FPP, attempt other assignment questions if I get stuck, and send questions at regular intervals.

7. Acknowledgements

Thanks to Prof Danvy for his patience.