coq-alpha-pearl README

Joshua Grosso
Faculty Mentor: Michael Vanier

September 5, 2021

1 Introduction

A Coq formalization of "Functional Pearls: α-conversion is easy" (Altenkirch, 2002). The most important file is `theories/Alpha.v`.

To our knowledge, all of the main results contained within the paper itself have been formalized. (Proving the equivalence to de Bruijn terms is still in progress, but because it is left to the reader in the original paper, we are comfortable sharing this formalization as-is.)

2 Technical Details

2.1 Project Setup

Install the `math-comp` and `extructures` libraries, and then run `make html` to build the project and its documentation (which annotates important definitions with their counterparts in the original paper).

2.2 Project Structure

The project is organized on disk as follows:

`theories/Alpha.v` contains the material from the original paper. It is parameterized by the infinite set $\mathcal{V}$ and its corresponding Fresh function.

`theories/AlphaImpl.v` contains two concrete implementations of the formalization, one where $\mathcal{V}$ is the set of finite strings and another where $\mathcal{V} = \mathbb{N}$. We expect the former to be the default choice for practical purposes; we implemented the latter merely as a curiosity.

`theories/Examples.v` contains several examples of $\equiv_\alpha$ in action.

`theories/Util.v` and `theories/Util/*.v` provide a small, custom utility library for the project. The one exception is `theories/Util/PlfTactics.v`, which is a copy of `LibTactics.v` from *Programming Language Foundations*.

2.3 External Libraries

The project relies on the `math-comp` library as a supplement to Coq's built-in standard library. It relies on `extructures` for extensionally-equal finite sets and maps. Finally, it relies on `LibTactics.v` from *Programming Language Foundations* (it exists as a copy in our project at `theories/Util/PlfTactics.v`).

3 Formalization Overview

This formalization contains several deviations from the original paper. Some were for ease of formalization, while others were (as best we can tell) required to prove the desired statements. We tried to be generally faithful to the provided proofs, at least for the more complex theorems. Finally, we admit the possibility that any or all of the below changes are misinterpretations on our part of the original author's meaning.

3.1 Typed Lambda Calculus

Our representation of the lambda calculus is untyped, for convenience. However, because the material is agnostic to the presence of types, it should be trivial to adapt it for e.g. the simply-typed λ-calculus.

3.2 (In)finite Sets

We encode the infinite set $\mathcal{V}$ as a type with an order:

```
Parameter V : ordType.
```

(Note that defining $\mathcal{V}$ as an `ordType` provides decidable equality).

The ordering constraint is not strictly necessary for formalization (and is not present in the original paper), but it allows us to use the `extructures` library's implementation of extensionally-equal finite sets. This is very convenient for formalization, and we expect that orderings exist for all usual definitions of $\mathcal{V}$ (e.g. the set of all finite strings, or $\mathbb{N}$). Of course, if that expectation proves to be incorrect, we could remove this constraint (for example, we could represent finite sets as linked lists with an appropriate equivalence relation, which we would manually use in lieu of definitional equality).

3.3 Finite Maps

We define partial bijections via the `extructures` library's implementation of extensionally-equal finite maps. We define substitutions similarly; we did not directly encode them as functions because we found that finite maps were more convenient (given that we are avoiding the use of classical logic).

3.4 Inductive Definitions as Boolean Predicates

Where possible, we preferred boolean predicates to inductive propositions (being sure, however, to prove equivalence with the original definitions). For example, consider the definition of $\mathrm{Tm}(X)$ (page 2). A straightforward formalization uses an inductive proposition, like so:

```
(* [x ∈ X] is notation for SSReflect's [x \in X]. *)
Section in_Tm.
  #[local] Reserved Notation "t '∈' 'Tm' X" (at level 40).

  Inductive in_Tm : {fset V} -> term -> Prop :=
  | Tm_variable : forall X x,
    x \in X ->
    variable x ∈ Tm X
  | Tm_application : forall X t u,
    t \in Tm X -> u \in Tm X ->
    application t u \in Tm X
  | Tm_abstraction : forall X t x,
    t \in Tm (X ∪ {x}) ->
    abstraction x t \in Tm X

  where "t '∈' 'Tm' X" := (in_Tm X t).
End in_Tm.
```

However, we can alternately define Tm as a boolean predicate (assuming suitable definitions of FV and $\subseteq$):

```
Definition Tm X t : bool := FV t ⊆ X.
```

Although it is admittedly subjective, we find the latter definition to be semantically clearer. We have proven that these two representations are equivalent:

```
Lemma TmP : forall X t, reflect (in_Tm X t) (t ∈ Tm X).
```

Similarly, we define, use, and prove correctness of algorithmic forms of $\equiv_\alpha^R$ and $\mathrm{Tm}^{\mathrm{db}}$. This makes them easy to use in runtime computations.

3.5 Explicitly-Provided Sets

When the original paper introduces a term $t \in \mathrm{Tm}(X)$, we note a slight ambiguity: Given that $\forall X, Y : X \subseteq Y \implies \mathrm{Tm}(X) \subseteq \mathrm{Tm}(Y)$, it is possible that $\mathrm{FV}(t) \subset X$ and thus X is not uniquely specified. For ease of formalization (and, to our knowledge, without loss of generality), we have largely replaced references to these sets with $\mathrm{FV}(t)$. For example, we have formalized Proposition 2.1 as $\forall t : t \equiv_\alpha^{1_{\mathrm{FV}(t)}} t$ instead of $\forall t \in \mathrm{Tm}(X) : t \equiv_\alpha^{1_X} t$. For another example,

we defined $t \equiv_\alpha u = t \equiv_\alpha^{1_{\mathrm{FV}(t)}} u$ (noting that $\mathrm{FV}(t) = \mathrm{FV}(u)$ by the assumed α-equivalence of t and u).

We employ a similar strategy for function domains and codomains. Instead of referencing the set X in $f \in X \longrightarrow \mathrm{Tm}(Y)$ (page 4), we instead reference $\mathrm{dom}(f)$. Similarly, instead of referencing Y, we calculate the smallest such set by iterating over the codomain of f (see `codomm_Tm_set`).

Our original approach had been to name these sets explicitly and take them as parameters to the propositions. We were successful in proving the main results of the paper, but the resulting definitions were very unwieldy. Eventually, we revised our work to calculate all such sets where possible, and found that the proof scripts became much shorter as a result.

3.6 Symmetric Updates

Assuming the original paper uses ordered tuples, we have changed the definition of $R(x, y) = (R \setminus \{(x, v), (y, w) | v, w \in \mathcal{V}\}) \cup \{(x, y)\}$ to $R(x, y) = (R \setminus \{(x, v), (w, y) | v, w \in \mathcal{V}\}) \cup \{(x, y)\}$. We believe this to have been the intention of the original author (given the emphasis on the symmetricality of the update procedure).

3.7 Lemma 1.3

We believe that the original proof for Lemma 1.3 (page 3) has a counterexample, arising from the second and third steps of the $\iff$ chain. Specifically, consider the backwards implication portion of that step. Let us separately consider the two cases of our hypothesis $(a = x \wedge z = b) \vee (x \neq a \wedge z \neq b \wedge aR; Sb)$. If $a = x$ and $z = b$, then we can let $c = y$ and our conclusion is satisfied. Thus, we will now assume instead that $x \neq a$, $z \neq b$, and $aR; Sb$. Because R and S are both partial bijections, there must exist a unique c for which aRc and cSb. However, as best we can tell, $c = y$ is not prohibited by the assumptions of the lemma; thus, for the sake of argument, we will assume this equality. In this case, neither case of the conclusion is possible, and so we have a contradiction.

In our formalization, we have replaced $R(x, y); S(y, z) = (R; S)(x, z)$ with $\forall a, b \in \mathcal{V} : aR(x, y); S(y, z)b \implies a(R; S)(x, z)b$. Fortunately, this is still strong enough to prove proposition 2.3, i.e. where lemma 1.3 was originally used.

3.8 Equivalence Classes

Coq does not yet support quotient types (at least not without additional axioms, which we were hesitant to add). Thus, instead of defining $Tm(X)/\equiv_\alpha$ at the term-level, we manually proved that the subsequent theorems respect $\equiv_\alpha$.

3.9 Substitution Extension

In the definition of $(\!|f|\!)$ (page 4), we were unclear whether or not Y is updated upon structural recursion, i.e. whether we should always reference the codomain

of the top-level f, or if we should instead use $\mathrm{cod}(f[x,z])$ when recursing in $(\!|f|\!)(\lambda x.t) = \lambda z.(\!|f[x,z]|\!)(t)$. We assumed the latter, and were able to prove the subsequent theorems.

3.10 Substitution Monad Laws

We found it necessary to add an additional condition $x \notin \mathrm{FV}(v)$ to the second monad law for substitution, a la Exercise 2.2 in `http://www.cse.chalmers.se/research/group/logic/TypesSS05/Extra/geuvers.pdf`.

3.11 Lemma 7

We instead proved $(\!|f|\!)(t) \equiv_\alpha^{f^\circ} t$; we assume this was the original author's intention.

3.12 Minor Textual Notes

On page 2, we presume that $X \subseteq_{\mathrm{fin}} \mathcal{V}$ is equivalent (by the infiniteness of $\mathcal{V}$) to $X \subset_{\mathrm{fin}} \mathcal{V}$.

On page 3, we presume that $R(x,y)\cdots \in (X \cup \{x\}) \times (Y \cup \{y\})$ uses $\in$ to represent $\subseteq$.

On page 5, we assume that "Hence by ind.hyp. we know $(\!|f[x,z]|\!)(t) \equiv_\alpha^{S(z,z')}$ $(\!|f[y,z']|\!)(u)$…" was intended to read "Hence by ind.hyp. we know $(\!|f[x,z]|\!)(t) \equiv_\alpha^{S(z,z')}$ $(\!|g[y,z']|\!)(u)$…".