Notes on the Lambek Eequivalence

Artem Gureev

Abstract

A summary of Sections 10, 11 of *Introduction to Higher Order Categorical Logic* by Lambek and Scott for the internal use by the Heliax team. The outline proceeds by dropping the natural numbers type in the definition of `STLC`.

The document serves as a technical showcase of one of the core motivational results of the Geb programming language. Namely, stating that good compilation procedures up to definitional equality of terms can be instead represented by cartesian closed functors between appropriate categories. The implementation of these results may help in a more intuitive construction of novel compilers as well as in formal verification of compiler procedures building on avid category theory formalization libraries already available.

By "a collection of types/terms" etc we mean "a collection of types/terms of simply typed λ-calculus." The notation for objects of `STLC` is changed to a more syntactic presentation to avoid confusion with the usual category theoretic notation. Formal definition of free variables, term formation by substitution and their preservation by translations added. Square brackets are used identically to round brackets with the purpose of making the notation more readable.

Contents

1 Simply Typed λ-calculus

Definition. *A **collection of types** is a set Ty such that*

1) $Unit \in Ty$

2) if $A, B \in Ty$ then $Prod(A, B) \in Ty$ and $Exp(B, A) \in Ty$.

Elements of Ty are called types.

Definition. *Given a collection of types Ty a **collection of terms** is an assignment $Tm : Ty \to Set$ such that for any $A, B \in Ty$*

1) $\{x_i^A\}_{\mathbb{N}} \subseteq Tm(A)$.

2) $* \in Tm(Unit)$

3) given $a \in Tm(A)$, $b \in Tm(B)$, $c \in Tm(Prod(A, B))$ we have

$$\begin{aligned} pair(a, b) &\in Ty(A \times B) \\ pr_1(c) &\in A \\ pr_2(c) &\in B \end{aligned}$$

4) given $f \in Tm(Exp(B, A))$ and $a \in Tm(a)$ we have $app(f, a) \in Tm(B)$

5) given x_i^A and $b(x_i^A) \in Tm(B)$ we have

$$\lambda x_i^A.b(x_i^A) \in Exp(B, A)$$

Given $A \in Ty$ and $a \in Tm(A)$ we say $a : A$, i.e. a is a term of type A

The notation $b(x_i^A)$ is there simply to showcase the dependency (possibly trivial). Yet as we have not established the definition of dependency, one may not worry about the brackets. In particular, say $b = b(x_i^A)$ if we can prove that $b : B$ given $x_i^A : A$

Definition. *Given $a : A$ a set of **free variables** of a, $FV(A)$ is defined inductively:*

1) $FV(x_i^A) := \{x_i^A\}$ for any $i \in \mathbb{N}$

2) $FV(*) := \emptyset$

3) $FV(pair(a, b)) := FV(a) \cup FV(b)$

4) $FV(pr_1(c)) = FV(pr_2(c)) := FV(c)$

5) $FV(\lambda x_i^A.b(x_i^A)) := FV(b(x_i^A)) - \{x_i^A\}$

6) $FV[app(f, a)] := FV(f) \cup FV(a)$

Given different λ-calculi, one has to explicitly specify their free variables

Definition. *A **set of equalities** for $A \in Ty$ is a set Eq_A of expressions of form*

$$X \vdash a1 \equiv_A a2$$

with X a finite set of variables of A satisfying:

1) $FV(a1) \cup FV(a2) \subseteq X$

2) $X \vdash \cdot \equiv_A \cdot$ *is an equivalence relation*

3) $a1, a2 : A$*,* $f : Exp(B, A)$*,* $X \vdash a_1 \equiv_A a2$ *then*

$$X \vdash app(f, a1) \equiv app(f, a2)$$

and $X \vdash b1(x_i^A) \equiv b2(x_i^A)$ *gives*

$$X - \{x_i^A\} \vdash \lambda x_i^A.b1(x_i^A) \equiv \lambda x_i^A.b2(x_i^A)$$

4) if $X \vdash a1 \equiv_A a2$ *and* $X \subseteq Y$ *then*

$$Y \vdash a1 \equiv_A a2$$

5) for $a : Unit$*, have*

$$FV(a) \vdash a \equiv_{Unit} *$$

6) for $a : A, b : B$ *have*

$$FV(a) \cup FV(b) \vdash pr_1[pair(a, b)] \equiv_A a$$
$$FV(a) \cup FV(b) \vdash pr_2[pair(a, b)] \equiv_B b$$

7) for $c : Prod(A, B)$ *have* $FV(c) \vdash pair(pr_1(c), pr_2(c)) \equiv_{Prod(A,B)} c$

8) for $f \in Exp(B, A)$*,* $x_i^A \notin FV(f)$

$$FV(f) \vdash \lambda x_i^A[app(f, x_i^A)] \equiv_{Exp(B^A)} f$$

We drop the subscript on the equivalence relation when evident for readability.

We also define substitution internally by taking $app[\lambda x_i^A.b(x_i^A), a] := b(a)$. We will later define morphisms of `STLC` so that they preserve type introductions and eliminations and hence will preserve substitution rules via the definition given in this paragraph.

Definition. *A **simply typed** λ**-calculus** is a 3-tuple* (Ty, Tm, Eq) *such that* Ty *is a collection of types,* $Tm, Eq : Ty \to Set$ *are collections of terms and collections of equalities pointwise.*

Definition. *A **translation** of a simply typed* λ*-calculus* $L_1 = (Ty_1, Tm_1, Eq_1)$ *to* $L_2 = (Ty_2, Tm_2, Eq_2)$ *is a function* $T_{ty} : Ty_1 \to Ty_2$ *alongside with a collection of functions* $T_{tm}(A) : Tm(A) \to Tm(T_{ty}(A))$ *for every* $A \in Ty_1$ *such that*

1) T_{ty} *strictly preserves type structure:*

$$T_{ty}(Unit) = Unit$$
$$T_{ty}[Prod(A, B)] = Prod[T_{ty}(A), T_{ty}(B)]$$
$$T_{ty}[Exp(B, A)] = Exp[T_{ty}(B), T_{ty}(A)]$$

2) given $a : A$*,* $T_{tm}(a) : T_{ty}(A)$

3) given $A \in Ty_1$

$$T_{tm}(x_i^A) = x_i^{T_{ty}(A)}$$

3) given $a : A$ *such that* $FV(a) = \emptyset$*, then* $FV[T_{tm}(a)] = \emptyset$

4) T_{tm} weakly preserves term-introduction and elimination, i.e.

$$
\begin{aligned}
&X \vdash T_{tm}[pair(a,b)] \equiv pair[T_{tm}(a), T_{tm}(b)]\\
&X \vdash T_{tm}[pr_1(c)] \equiv pr_1[T_{tm}(c)]\\
&X \vdash T_{tm}[pr_2(c)] \equiv pr_2[T_{tm}(c)]\\
&X \vdash T_{tm}[\lambda x_i^A.b(x_i^A)] \equiv \lambda x_i^{T_{ty}(A)}.T_{tm}[b(x_i^A)](x_i^{T_{ty}(A)})\\
&X \vdash T_{tm}[app(f,a)] \equiv app[T_{tm}(f), T_{tm}(a)]
\end{aligned}
$$

where X is a set of free variables of the corresponding terms

5) if $X \vdash a \equiv b$ then

$$\bigcup_{x \in X} T_{tm}(x) \vdash T_{ty}(a) \equiv T_{ty}(b)$$

note that the last axiom allows for 4 to exclude the need to check $* : Unit$. Moreover, it is sufficient that T_{ty} preserved equalities on all elements of Eq_1 to check condition 5

We use the notation $(T_{ty}, T_{tm}) : L_1 \to L_2$

Note that Eq is pointwise thought of as sets of **definitional** equalities while still being formal symbols. We want two compilation procedures who end up being provably definitionally equal pointwise to be treated as same. Hence we introduce the quotienting relation:

Definition. *Translations $T^1, T^2 : L_1 \to L_2$ are **equivalent** if given $X \vdash a \equiv b$ we have that*

$$\bigcup_{x \in X} T^1_{tm}(x) \vdash T^1_{tm}(a) \equiv T^2_{tm}(b)$$

Proposition. *The relation of being equivalent translations is an equivalence relation*

Definition. `STLC` *is a category whose objects are simply typed lambda calculi and morphisms translations modulo equivalence*

2 Cartesian Closed Categories

Definition. *A category C is cartesian closed if*

1) It has all finite products

2) For every $A \in C$ we have a right adjoint to the product:

$$\begin{tikzcd} C \arrow[r, bend right, "{[A,(-)]}"'] \arrow[r, phantom, "\perp"] & C \arrow[l, bend right, "(-)\times A"'] \end{tikzcd}$$

Definition. *Given objects X, Y in a category C, an exponential object $X^Y \in C$ is an object with a map $eval : X^Y \times Y \to X$ such that for every Z and $f \colon Z \times Y \to X$ we have*

$$\begin{tikzcd} Z \arrow[d, "\exists! \lambda f"'] \\ X^Y \end{tikzcd} \qquad \begin{tikzcd} Z \times Y \arrow[d, "\lambda f \times 1_Y"'] \arrow[dr, "f"] & \\ X^Y \times Y \arrow[r, "eval"'] & Y \end{tikzcd}$$

Proposition. *If C is a category with finite products, it is cartesian closed if and only if it has all exponential objects*

Proof. The constructions go as follows:

If the category is cartesian closed, then let *eval* be a function corresponding to the image of the identity on X^Y under the natural iso $C([Y,X] \times Y, X) \cong C([Y,X],[Y,X])$. λ operation is given by the natural isomorphism as well considering the needed naturality square.

Given Y, $(-)^Y$ is a functor taking X to X^Y and $f : X \to Z$ to $\lambda(f \circ eval) \colon X^Y \to Z^Y$, the unique map filling

$$\begin{tikzcd} X^Y \times Y \arrow[dd, "\lambda(f \circ [eval_X]) \times 1_Y"'] \arrow[dr, "eval_X"] & & \\ & X \arrow[dr, "f"] & \\ Z^Y \times Y \arrow[rr, "eval_Z"'] & & Z \end{tikzcd}$$

this will be the right adjoint.

The natural equivalence $C(X \times Y, Z) \cong C(X, Z^Y)$ is given by sending $f \in C(X \times Y, Z)$ to λf and $g \in C(X, Z^Y)$ to $eval \circ (g \times 1_Y)$ $\square$

Definition. *A functor $F \colon C \to D$ between cartesian closed categories is **cartesian closed** if it strictly preserves the structure of products (including terminal) and exponential objects:*

1) $F(1) = 1$

2) $F(A \times B) = F(A) \times F(B)$

3) $F(\pi_i)$, the projection from $F(A) \times F(B)$ is π_i in D for $i = 1, 2$ [i.e. the legs of limits are preserved]

4) $F(X^Y) = F(X)^{F(Y)}$

5) $F(eval) = eval$

This is sufficient to show that all the structure stated in Lambek-Scott is preserved. E.g. given $f\colon Z \to X$, $g\colon Z \to Y$, $F(\langle f, g\rangle) = \langle F(f), F(g)\rangle$ which we can check by using the fact that limit legs are jointly monic:

$$\begin{aligned}\pi_1 \circ F(\langle f, g\rangle) &= F(\pi_1) \circ F(\langle f, g\rangle)\\ &= F(\pi_1 \circ \langle f, g\rangle)\\ &= F(f)\end{aligned}$$

similarly on the other side.

For simplicity the isomorphism $C(X \times Y, Z) \to C(X, Z^Y)$ is called *curry*.

Definition. `CCC` *is a category whose objects are cartesian closed categories and morphisms cartesian closed functors.*

3 Internal Language Theorem for CCC

The result of Lambek tells us that given a simply typed λ-calculus L we can construct a cartesian closed category, whose objects are types of L, with $Unit, Pair(A, B), Exp(B, A)$ - serving as $1, A \times B, B^A$ - and morphisms 2-tuples, consisting of a variable in the domain and a term in the codomain built **solely** out of the mentioned variable.

The constructions turns out be an equivalence of categories. Namely, one can either work in `STLC` or `CCC` given the interest in either cartesian closed functors or compilation procedures preserving unit, pairing, and exponentiation.

3.1 Syntactical Category of a Simply Typed λ-calculus

Definition. *A functor Syn : `STLC` $\rightarrow$ `CCC` taking a simply typed λ-calculi to corresponding syntactical categories is given by the following data:*

Given a calculus (Ty, Tm, Eq)

1) The set of objects of $Syn(Ty, Tm, Eq)$ is Ty

2) Given $A, B \in Ty$,

$$Hom(A, B) := \left\{(x_i^A, b(x_i^A) \mid FV(b(x_i^A)) \subseteq \{x_i^A\}\right\} \sim$$

where $\sim$ is an equivalence relation defined as:

$(x_i^A, b1(x_i^A)) \sim (x_j^A, b2(x_j^A))$ if and only if $\{x_k^A\} \vdash b1(x_k^A) \equiv b2(x_k^A)$

3) For $A \in Ty$, $1_A := (x_i^A, x_i^A)$

4) Given $(x_i^A, b(x_i^A))\colon A \rightarrow B$, $(x_j^B, c(x_j^B))\colon B \rightarrow C$

$$(x_j^B, c(x_j^B)) \circ (x_i^A, b(x_i^A)) := (x_i^A, c(b(x_i^A))$$

Cartesian closed structure is granted by:

1) $1 := Unit$, $!_A := (x_i^A, *)$

2) $A \times B := Pair(A, B)$, $\pi_i := (x_j^{Pair(A,B)}, pr_i(x_j))$

3) $A^B := Exp(A, B)$, $eval := (x_i : Pair[Exp(A, B), C], app[pr_1(x), pr_2(x)])$

Given a translation $T : L_1 \rightarrow L_2$, define $Syn([T])$ to be a functor such that:

1) $Syn([T])(A) := T_{ty}(A)$

2) $Syn([T])((x_i^A, b(x_i^A)) := (x_i^{T_{ty}(A)}, T_{tm}[b(x_i^A)])$

Firstly, this is well-defined as $[T]$ is defined up to definitional equality of terms.

Moreover, note that this is indeed a functor. Identities are preserved as variables are preserved. Composition is preserved as substitution is preserved by translations.

Syn sends the identity translation to the identity as its function on types and terms are the identity. Composition is also easily checked to be preserved.

3.2 Internal Language of a Cartesian Closed Category

Warning: this section uses more advanced categorical logic one may not be familiar with, but the results make precise what we need to do in order to construct an internal language of a cartesian closed category. It also may deviate from the original presentation of the proof. All faults of the proof are due to the author of the notes.

Given a category C, we can express all of its information relating to cartesian closedness equationally. Category theory is an essentially algebraic theory. Now add to it axioms stating that Obj_C has all the objects of C and expressing that coproducts and exponentials have the desired universal properties. Moreover, add $x_i \colon 1 \to A$ for every $A \in C$, $i \in \mathbb{N}$. Name this theory EAT_C. Being an essentially algebraic theory, this will have an initial model.

Definition. *Given a cartesian category C, define C_{var} to be the intitial model of the theory EAT_C.*

To be an initial model means to be a category which is freely built out of the objects and terms in language modulo equality axioms. For a more hand-on construction, consider the following pseudocode in Agda syntax:

```
record CC-Category : Set where
  field
    Cat : Category
    [_,_] : Obj Cat -> Obj Cat -> Obj Cat
    _x_ : Obj Cat -> Obj Cat -> Obj Cat
    1 : Obj Cat
    exp-prf : {A B : Obj Cat} -> is-exponential [ A , B ]
    -- proof that [ A, B ] is an internal hom for every A, B

    prod-prf : {A B : Obj Cat} -> is-product (A x B)
    -- proof that A x B is a product for every A , B

    terminal-prf : is-terminal 1
    -- proof that 1 is a terminal object

postulate
  -- postulate some cartesian closed category

  C : CC-Category

data _-var (C : CC-Category) (A B : Obj C) : Set where
  -- postulate the set of morphisms by freely adding variables while
     not changing the objects

  var : {A : Obj C} {n : \bN} -> C-var 1 A
  inmor :  {A B : Obj C} {f : (Hom C) A B} -> C-var A B
  comp-var : {A B C : Obj C} {f : C-var B C} {g : C-var A B}
                                                   -> C-var A C

postulate
  -- add an axiom saying that composition for morphisms from C is
```

```
  the same as before

compeq : {C : CC-Category} {A B C : Obj C} (f : (Hom C) B C)
                                          (g : (Hom C) A B) ->
                   comp-var (inmor f) (inmor g) \== inmor (comp f g)
```

```
postulate
  C-var-cat : is-category C (C-var) comp-var
  -- axiom that C-var is a category with objects C, morphisms from
     C-var and composition comp-var

  C-var-ccc : is-ccc C-var-cat [_,_] _x_ 1
  -- axiom that C-var has the same objects having exponential, product,
     terminal object structure as C
```

Note the evident inclusion $C \hookrightarrow C_{var}$

Proposition. *Let CCC_{var} be a category of cartesian closed categories with a choice of countably many arrows $1 \to A$ which we call variables. We then have a free-forgetful*

$$\begin{tikzcd} CCC \arrow[r, bend left, "F"] \arrow[r, phantom, "\perp"] & CCC_{var} \arrow[l, bend left, "U"] \end{tikzcd}$$

Proof. Both CCC and CCC_{var} are categories which are equivalent to the categories of models of their underlying essentially algebraic theories. Moreover, the theory of CCC_{var} is just the theory of CCC with additional axioms stating the existence of countably many $1 \to A$ for every A in the category. That is, we have a forgetful interpretation of the theory of CCC_{var} in CCC. □

Proposition. $C_{var} \cong F(C)$

Proof. Suffices to show that both have the same universal property. Note that $F(C) = UF(C)$ and $\eta_C : 1_C \to UF(C)$ is the initial object in $C \downarrow U$

By Proposition 5.1 in Lambek-Scott, this is exactly the universal property of the inclusion $C \hookrightarrow C_{var}$ □

For more references on this topic, see Lambek-Scott Section 3, Adamek-Rosicky *Localy Presentable and Accessible Categories*

Definition. *A polynomial $f : 1 \to B$ is called dependent on $x : 1 \to A$ written $f(x)$ if it is in C_{var} but not in C.*

Note that for the purposes of giving an internal language we may write $f(x)$ for constant polys, i.e elements of C rather than C_{var}.

Note that the dependency may be trivial. I.e suppose we have a constant polynomial $b : 1 \to B$ and a variable $x : 1 \to A$. Then b is trivially dependent on x by the fact that $b = b \circ !_A \circ x$

Definition. *Given variable $x_i^A : 1 \to A$, we define $unvar_{A,i} : C_{var}(B, C) \to C(A \times B, C)$ by induction on the structure of morphisms:*

$$
\begin{aligned}
unvar(x_i^A) &:= \pi_1 \\
unvar(x_j^B) &:= x_j^B \circ \pi_1 \\
unvar(k) &:= k \circ \pi_1 && \text{for } k \in C \\
unvar(f \circ g) &:= unvar(f) \circ \langle \pi_1, unvar(g) \rangle \\
unvar(\langle f, g \rangle) &:= \langle unvar(f), unvar(g) \rangle \\
unvar(\lambda f) &:= \lambda(unvar(f))
\end{aligned}
$$

This corresponds to removing the dependency on x_i^A.

Definition. *A function Lang:* `CCC` $\to$ `STLC` *taking cartesian closed categories to the corresponding internal languages is defined as follows:*

Given a cartesian closed category C

1) $Ty = Obj(C)$, with $Unit := 1$, $Prod(A, B) := A \times B$, $Exp(B, A) := B^A$

2) for every A, $Ty(A) := C_{var}(1, A)$ with

$$
\begin{aligned}
x_i^A &:= x_i \\
* &:= !_1 \\
pair(a, b) &:= \langle a, b \rangle \\
pr_1(c) &:= \pi_1 \circ c \\
pr_2(c) &= \pi_2 \circ c \\
app(f, a) &= eval \circ \langle f, a \rangle \\
\lambda x_i^A.b(x_i^A) &:= curry[unvar_{A,i}(b(x_i^A)) \circ switch]
\end{aligned}
$$

3) The equalities are given by all the equalities satisfied in C_{var} where $X \vdash f \equiv_A g$ iff $f \equiv g$ and all variables in f and g appear in X. Equalities needed by STLCs are satisfied via universal properties of products and exponential object.

Given a cartesian closed functor $F : C \to D$ we have a unique

$$
\begin{tikzcd}
C_{var} \arrow[r, dashed, "\exists! F_{var}"] & D_{var} \\
C \arrow[u, hook] \arrow[r, "F"] & D \arrow[u, hook]
\end{tikzcd}
$$

which gives a translation $Lang(F) : Lang(C) \to Lang(D)$ by the object part of F_{var} on Types and F_{var} on morphisms for terms. Variables are preserved by the universal property of the unit, and equalities are preserved as functors preserve equalities.

Theorem. `STLC` $\simeq$ `CCC` *via the $Syn \dashv Lang$ adjunction*

4 Additional Remarks

The result can be expanded to the followng:

1) The category of simply typed λ-calculi with the natural number type is equivalent to the category of CCCs with weak natural number object.

2) A category of simply typed λ-calculi with plus-types are equivalent to bicartesian closed categories.

Geb without dependent types, in particular, corresponds to the initial object of the second construction.

Free variables, substitutions are treated loosely, functoriality proofs are also skipped. This document tries to reverse engineer the needed components to formally define all these. Generally, free variables ought to be treated even more systematically than here. One ought to have a set of type formers representing n-ary function symbols over types and form the corresponding sets of free variables similarly to how one does for algebraic theories or alternatively make all terms come in a set context of what we define to be free variables.

All the mistakes are due to the author of the notes.