Advances in Logical Grammar: Review of Typed Lambda Calculus

Carl Pollard

June 6, 2012

The Two Sides of Typed Lambda Calculus

A typed lambda calculus (TLC) can be viewed in two complementary ways:

- model-theoretically, as a system of notation for functions
- proof-theoretically, as an elaboration of natural deduction for intuitionistic propositional logic (IPL)
- In our linguistic application, we'll view it both ways simultaneously.
- But first, what *is* a TLC?

A TLC is a Lot Like a First-Order Logic

- A TLC has a lot in common with a FOL, starting with having both a syntax and a semantics.
- The syntax of a TLC has a lot in common with the syntax of a FOL, including constants, variables, variable binding, and rules for forming terms.
- The semantics of a TLC has a lot in common with the semantics of a FOL, including a class of set-theoretic interpretations and variable assignments.

What a TLC has that an FOL doesn't

- A FOL only has two types of terms: **individual** terms (often just called terms) and **truth-value** terms (often called **formulas**); whereas a TLC has an infinite number of types of terms, formed with **type constructors** by starting with a finite number of **basic** types.
- A TLC has the binding operator λ (lambda), which is the crucial ingredient for notating functions.

What a FOL has that a TLC doesn't

- A FOL has a special type of term – **truth value** terms (also called **formulas**) that can be used to express theories.
- A FOL has an **equality** symbol which can be used to form formulas (by placing it between two individual terms).
- A FOL has logical connectives and quantifiers for forming more complex formulas.

The Best of Both Worlds

- Before long, we'll see how to construct systems—**higher order logics (HOLs)** that combine all the features of TLCs and FOLs.
- We'll use one of these, the **pheno** logic, to notate (and theorize about) phenogrammar.
- We'll use another one, the **semantic** logic, to notate (and theorize about) meanings.
- the tecto linear logic makes three.
- When we analyze signs, we'll be doing proofs in all three of these logics, in parallel.

Specifying the Syntax of a TLC

1. We start by specifying the **basic types**.
2. We use the type constructors to recursively define the full set of types.
3. We specify a finite number of **constants** and assign each constant a type.
4. Finally, we use the term-forming rules to recursively define the full set of terms and assign each term a type.

As running examples, we'll go through this process for two different TLCs (one for pheno and one for semantics).

Basic Types

- In the simplest approach to pheno, the pheno TLC has just one basic type s (string). (Eventually it becomes necessary to add more basic pheno types, e.g. for phonological words, clitics, pitch accents, etc.).
- The semantic TLC has the two basic types e (entities, the meanings of (uses of) proper nouns), and p (propositions, the meanings of (uses of) declarative sentences).

Defining the Full Set of Types of a TLC

- T is a type.
- If A and B are types, then so are:
  - $A \to B$
  - $A \wedge B$
  - $A \vee B$
- Nothing else is a type (in particular, we don't make use of F, negation, or quantifiers).

Note: The set of types is the same as the set of IPL formulas obtained by taking the basic types to be the atomic formulas.

TLC Constants

Note: we write '$\vdash a : A$' to mean term a is of type A.

- Every TLC has the **logical** constant $\vdash * : \mathrm{T}$.
- Constants of the pheno TLC:

  $\vdash \mathbf{e} : \mathrm{s}$ (null string)

  $\vdash \cdot : \mathrm{s} \to \mathrm{s} \to \mathrm{s}$ (concatenation)

  Note: usually written infix, e.g. $s \cdot t$ for $(\cdot\ s\ t)$

  constants for strings of single phonological words, e.g. $\vdash$ pig : s for the string of /pIg/.

- Constants of the semantic TLC, e.g.

  $\vdash \mathsf{fido} : \mathrm{e}$

  $\vdash \mathsf{bark} : \mathrm{e} \to \mathrm{p}$

  $\vdash \mathsf{maybe} : \mathrm{p} \to \mathrm{p}$

  $\vdash \mathsf{bite} : \mathrm{e} \to \mathrm{e} \to \mathrm{p}$

  $\vdash \mathsf{give} : \mathrm{e} \to \mathrm{e} \to \mathrm{e} \to \mathrm{p}$

  $\vdash \mathsf{believe} : \mathrm{e} \to \mathrm{p} \to \mathrm{p}$

TLC Terms (1/2)

a. For each constant a of type A, $\vdash a : A$.

b. For each type A there are **variables** $\vdash {x_i}^A : A$ $(i \in \omega)$.

c. If $\vdash f : A \to B$ and $\vdash a : A$, then $\vdash \mathsf{app}(f, a) : B$.

   Note: $\mathsf{app}(f, a)$ is abbreviated to $(f\ a)$.

d. If $\vdash x : A$ is a variable and $\vdash b : B$, then $\vdash \lambda_x.b : A \to B$.

TLC Terms (2/2)

e. If $\vdash a : A \wedge B$, then $\vdash \pi(a) : A$.

f. If $\vdash a : A \wedge B$, then $\vdash \pi'(a) : B$.

g. If $\vdash a : A$ and $\vdash b : B$, then $\vdash (a, b) : A \wedge B$.

h. If $\vdash x : A$ and $\vdash y : B$ are variables, $\vdash d : A \vee B$, $\vdash c : C$, and $\vdash c' : C$, then $[\mathbf{case}\ d\ (\iota(x)\ c)\ (\iota'(y)\ c')] : C$.

i. If $\vdash a : A$, then $\vdash \iota_{A,B}(a) : A \vee B$

j. If $\vdash b : B$, then $\vdash \iota'_{A,B}(b) : A \vee B$

Note: subscripted A, B on π, π', ι, and ι' are suppressed for the sake of readability.

TLC Term Equivalences (1/3)

Here t, a, b, p, and f are metavariables ranging over terms.

a. Equivalences for the term constructors:

1. $t \equiv *$ (for t a term of type T)
2. $\pi(a, b) \equiv a$
3. $\pi'(a, b) \equiv b$
4. $(\pi(p), \pi'(p)) \equiv p$

TLC Term Equivalences (2/3)

b. Equivalences for the variable binder ('lambda conversion')

(α) $\lambda_x.b \equiv \lambda_y.[x/y]b$

(β) $(\lambda_x.b)\ a \equiv [x/a]b$

(η) $\lambda_x.(f\ x) \equiv f$, provided x is not free in f

Note 1: The notation '$[x/a]b$' means the term resulting from substitution in b of all free occurrences of $x : A$ by $a : A$. This presupposes a is free for x in b.

Note 2: 'Free' and 'bound' are defined just as in FOL, except that λ is the variable binder rather than $\forall$ and $\exists$.

TLC Term Equivalences (3/3)

c. Equivalences of Equational Reasoning

(ρ) $a \equiv a$

(σ) If $a \equiv a'$, then $a' \equiv a$.

(τ) If $a \equiv a'$ and $a' \equiv a''$, then $a \equiv a''$.

(ξ) If $b \equiv b'$, then $\lambda_x.b \equiv \lambda_x.b'$.

(μ) If $f \equiv f'$ and $a \equiv a'$, then $(f\ a) \equiv (f'\ a')$.

Set-Theoretic Interpretation of a TLC

A **(set-theoretic) interpretation** I of a TLC assigns to each type A a set $I(A)$ and to each constant $\vdash a : A$ a member $I(a)$ of $I(A)$, subject to the following constraints:

1. $I(\mathrm{T})$ is a singleton
2. $I(A \wedge B) = I(A) \times I(B)$
3. $I(A \vee B) = I(A) + I(B)$ (disjoint union)
4. $I(A \to B) \subseteq I(A) \to I(B)$

Note: The set inclusion in the last clause can be proper, as long as there are enough functions to interpret all terms.

Assignments

An **assignment** relative to an interpretation is a function that maps each variable to a member of the set that interprets that variable's type.

Extending an Interpretation Relative to an Assignment

Given an assignment α relative to an interpretation I, there is a unique extension of I, denoted by I_α, that assigns interpretations to all terms, such that:

1. for each variable x, $I_\alpha(x) = \alpha(x)$
2. for each constant a, $I_\alpha(a) = I(a)$
3. if $\vdash a : A$ and $\vdash b : B$, then $I_\alpha((a, b)) = \langle I_\alpha(a), I_\alpha(b)\rangle$
4. if $\vdash p : A \wedge B$, then $I_\alpha(\pi(p))$ = the first component of $I_\alpha(p)$; and $I_\alpha(\pi'(p))$ = the second component of $I_\alpha(p)$
5. if $\vdash f : A \to B$ and $\vdash a : A$, then $I_\alpha((f\ a)) = (I_\alpha(f))(I_\alpha(a))$
6. if $\vdash b : B$, then $I_\alpha(\lambda_{x \in A}.b)$ is the function from $I(A)$ to $I(B)$ that maps each $s \in I(A)$ to $I_\beta(b)$, where β is the assignment that coincides with α except that $\beta(x) = s$.

Observations about Interpretations

- Two terms $\vdash a : A$ and $\vdash b : B$ of TLC are term-equivalent iff $A = B$ and, for any intepretation I and any assignment α relative to I, $I_\alpha(a) = I_\alpha(b)$.
- Another way of stating the preceding is to say that term equivalence (viewed as an equational proof system) is sound and complete for the class of set-theoretic interpretations described earlier.
- For any term a, $I_\alpha(a)$ depends only on the restriction of α to the free variables of a.
- In particular, if a is a closed (i.e. has no free variables), then $I_\alpha(a)$ is independent of α so we can simply write $I(a)$.
- Thus, an interpretation for the basic types and constants extends uniquely to all types and all closed terms.

Sequent-Style ND with Proof Terms for IPL

- This is a style of ND designed to analyze not just provability, but also proofs.
- It is an elaboration of the sequent-style ND for IPL already introduced.
- We'll see that in addition to being thought of as denoting elements of models, TLC terms can also be thought of as notations for proofs.
- This idea was first articulated by Curry (1934, 1958), then elaborated by Howard (1969 [1980]), Tait (1967), etc..
- We'll use this kind of ND for phenos and meanings in linear grammar derivations.

Preliminary Definitions

1. A (TLC) term is called **closed** if it has no free variables.
2. A closed term is called a **combinator** if it contains no nonlogical constants.
3. A type is said to be **inhabited** if there is a closed term of that type.

Curry-Howard Correspondence (1/2)

- Types are (the same thing as) formulas.
- Type constructors are logical connectives.
- (Equivalence classes of) terms are proofs.

- The free variables of a term are the undischarged hypotheses on which the proof depends.
- The nonlogical constants of a term are the nonlogical axioms used in the proof.
- A type is a theorem iff it is inhabited.
- A type is a pure theorem (requires no nonlogical axioms to prove it) iff it is inhabited by a combinator.

Curry-Howard Correspondence (2/2)

- Application corresponds to Modus Ponens.
- Abstraction corresponds to Hypothetical Proof (discharge of hypothesis).
- Pairing corresponds to Conjunction Introduction.
- Projections correspond to Conjunction Eliminations.
- Case corresponds to Disjunction Elimination.
- Canoniical injections correspond to Disjunction Introductions
- Identification of free variables corresponds to collapsing of duplicate hypotheses (Contraction).
- Vacuous abstraction corresponds to discharge of a nonexistent hypothesis (Weakening).

Notation for Sequent-Style ND with Proof Terms

Judgments are of the form $\Gamma \vdash a : A$, read 'a is a proof of A with hypotheses Γ', where

1. A is a formula (= type)
2. a is a term (= proof)
3. Γ, the **context** of the judgment, is a set of variable/formula pairs of the form $x : A$, with a distinct variable in each pair.

Axiom Schemas

Hypotheses:

$$x : A \vdash x : A$$

(x a variable of type A)

Nonlogical Axioms:

$$\vdash a : A$$

(a a nonlogical constant of type A)

Logical Axiom:

$$\vdash * : \mathrm{T}$$

Rule Schemas for Implication

→-Elimination or Modus Ponens:

$$\frac{\Gamma \vdash f : A \to B \qquad \Delta \vdash a : A}{\Gamma, \Delta \vdash (f\ a) : B} \to\mathrm{E}$$

This presupposes no variable occurs in both Γ and Δ.

→-Introduction or Hypothetical Proof:

$$\frac{x : A, \Gamma \vdash b : B}{\Gamma \vdash \lambda_x.b : A \to B} \to\mathrm{I}$$

Other Rule Schemas

There are also schemas (which we will introduce as needed) for:

- pairing/conjunction introduction
- projections/conjunction elimination
- case/disjunction elimination
- canonical injections/disjunction introduction
- identifying variables/contraction
- useless hypotheses/weakening