Simply Typed λ-Calculus

We begin with a quick review of the λ-calculus, culminating in a proof theory for the calculus.

Recall that at the core of the typed λ-calculus is a set of types:

> From a nonemtpy set **BasTyp** of basic types, the set **Typ** or types is the smallest set such that:
>
> a. $\mathbf{BasTyp} \subseteq \mathbf{Typ}$;
>
> b. if $\sigma, \tau \in \mathbf{Typ}$ then $(\sigma \to \tau) \in \mathbf{Typ}$.

Following Montague, let's take

$$\mathbf{BasTyp} = \{\mathbf{Ind}, \mathbf{Bool}\}$$

where **Ind** is the type of individuals and **Bool** is the type of propositions.

Note that there are other kinds of typing systems (e.g., parametric types and type inheritance systems), but simply typed systems have the clearest structure and have been most widely applied to natural languages.

Recall that in order to construct λ-terms we first need to define:

a. $\mathbf{Var}_\tau$: a countably infinite set of variables of type τ.

b. $\mathbf{Con}_\tau$: a set of constants of type τ.

c. $\mathbf{Var} = \cup_{\tau \in \mathbf{Typ}} \mathbf{Var}_\tau$

d. $\mathbf{Con} = \cup_{\tau \in \mathbf{Typ}} \mathbf{Con}_\tau$

We can now define λ-terms in terms of a collection of sets, $\mathbf{Term}_\tau$, the smallest sets such that:

a. $\mathbf{Var}_\tau \subseteq \mathbf{Term}_\tau$;

b. $\mathbf{Con}_\tau \subseteq \mathbf{Term}_\tau$;

c. if $\alpha \in \mathbf{Term}_{\sigma \to \tau}$ and $\beta \in \mathbf{Term}_\sigma$, then $\alpha(\beta) \in \mathbf{Term}_\tau$;

d. $\lambda x.(\alpha) \in \mathbf{Term}_\tau$ if $\tau = \sigma \to \rho$ and $x \in \mathrm{Var}_\sigma$ and $\alpha \in \mathbf{Term}_\rho$.

In order to give a semantics for λ-terms, we need to define notions like substitution which, in turn, rest on the definition of *free* and *bound* variables:

> The set $\mathrm{Free}(\alpha)$ of free variables of the λ-term α is defined by:
>
> a. $\mathrm{Free}(x) = \{x\}$ if $x \in \mathbf{Var}$;
>
> b. $\mathrm{Free}(c) = \emptyset$ if $c \in \mathbf{Con}$;
>
> c. $\mathrm{Free}(\alpha(\beta)) = \mathrm{Free}(\alpha) \cup \mathrm{Free}(\beta)$,
>
> d. $\mathrm{Free}(\lambda x.\alpha) = \mathrm{Free}(\alpha) - \{x\}$.
>
> A variable that is not free is *bound*.

We turn next to substitution of one term for another:

> The result $\alpha[x \mapsto \beta]$ of the substitution of β for the free occurences of x in α is defined by:
>
> a. $x[x \mapsto \beta] = \beta$;
>
> b. $y[x \mapsto \beta] = y$ if $y \in \mathbf{Var}, x \neq y$;
>
> c. $c[x \mapsto \beta] = c$ if $c \in \mathbf{Con}$;
>
> d. $\alpha(\gamma)[x \mapsto \beta] = \alpha[x \mapsto \beta](\gamma[x \mapsto \beta])$;
>
> e. $(\lambda x.\alpha)[x \mapsto \beta] = \lambda x.\alpha$;
>
> f. $(\lambda y.\alpha)[x \mapsto \beta] = \lambda y.(\alpha[x \mapsto \beta])$ if $x \neq y$.

Finally, we define what it means for a term to be *free for substitution*:

> The term α is *free for* x in β, FreeFor(α, x, β), if and only if one of the following holds:
>
> a. $\beta \in \mathbf{Con}$;
>
> b. $\beta \in \mathbf{Var}$;
>
> c. $\beta = \gamma(\delta)$ and FreeFor(α, x, γ) and FreeFor(α, x, δ);
>
> d. $\beta = \lambda y.\gamma$ and FreeFor(α, x, γ) and either $x \notin$ Free(γ) or $y \notin$ Free(α).

The definition appears somewhat complicated but actually combines parts of $\beta[x \mapsto \alpha]$ and Free. Thus, we can say that $\mathbf{float}(y)$ is free for x in $\lambda z.z(x)$ since we can merge the two expressions to $\lambda z.z(\mathbf{float}(y))$ without accidently binding anything.

We turn now to providing models for the simply typed λ-calculus. This is a very straightforward matter which proceeds by first defining the domains for the types and then an interpretation function $[\![\cdot]\!]$.

> A frame for the collection $\mathbf{BasTyp}$ consists of a collection $\mathbf{Dom} = \cup_{\tau \in \mathbf{BasTyp}} \mathbf{Dom}_\tau$ of basic domains. Domains for functional types are defined by:
>
> $$\mathbf{Dom}_{\alpha\to\beta} = \mathbf{Dom}_\beta^{\mathbf{Dom}\alpha} = \{f \,|\, f : \mathbf{Dom}_\alpha \to \mathbf{Dom}_\beta\}$$

Having defined frames, we can turn to the definition of a model for the simply typed λ-calculus:

> A model of the simply typed λ-calculus is a pair $\mathcal{M} = \langle \mathbf{Dom}, [\![\cdot]\!] \rangle$ in which:
>
> a. $\mathbf{Dom}$ is a frame;
>
> b. the interpretation function $[\![\cdot]\!] : \mathbf{Con} \to \mathbf{Dom}$ respects typing so that $[\![\alpha]\!] \in \mathbf{Dom}_\tau$ if $\alpha \in \mathbf{Con}_\tau$.

Missing from the above, of course, is the notion of assignment:

$$\theta : \mathbf{Var} \to \mathbf{Dom}$$

such that $\theta(x) \in \mathbf{Dom}_\tau$ if $x \in \mathbf{Var}_\tau$.

We can now provide a recursive definition of the denotations of λ-terms:

The denotation $[\![\alpha]\!]^\theta_{\mathcal{M}}$ of a term α withe respect to the model $\mathcal{M} = \langle \mathbf{Dom}, [\![\cdot]\!] \rangle$ and assignment θ is given by:

a. $[\![x]\!]^\theta_{\mathcal{M}} = \theta(x)$ if $x \in \mathbf{Var}$;
b. $[\![c]\!]^\theta_{\mathcal{M}} = [\![c]\!]$ if $c \in \mathbf{Con}$;
c. $[\![\alpha(\beta)]\!]^\theta_{\mathcal{M}} = [\![\alpha]\!]^\theta_{\mathcal{M}}([\![\beta]\!]^\theta_{\mathcal{M}})$;
d. $[\![\lambda x.(\alpha)]\!]^\theta_{\mathcal{M}} = f$ such that $f(a) = [\![\alpha]\!]^{\theta[x:=a]}_{\mathcal{M}}$.

where $\theta[x := a]$ is that assignment that maps x to a and maps $y \neq x$ to whatever θ maps y to.

Theorem: Type Soundness
If α is a term of type τ, then $[\![\alpha]\!]^{\theta}_{\mathcal{M}} \in \mathbf{Dom}_{\tau}$ for every model $\mathcal{M}$ and assignment θ.

Proof Sketch: The proof is by induction on terms. The base case, on constants and variables, is obviously correct. One needs only extend the induction to:

i. function application;

ii. λ-terms.

There are a number of properties of this system that are quite equivalent to simple first order logic:

1. The identity of bound variables is irrelevant.
2. The notion of logical equivalence is easily defined:

   > Two λ-terms α and β are logically equivalent, $\alpha \equiv \beta$, if and only if $[\![\alpha]\!]^{\theta}_{\mathcal{M}} = [\![\beta]\!]^{\theta}_{\mathcal{M}}$ for every model $\mathcal{M}$ and assignment θ.

Furthermore, there are equivalences related to function application; in particular, the λ calculus forces function application to one argument at a time.

We can devise a method to show that this is a mere matter of notation and not deep logical property. For any type, $(\sigma \to (\tau \to \rho))$ we can define:

$$\mathbf{perm}_{\sigma,\tau,\rho} \stackrel{\text{def}}{=} \lambda f^{(\sigma\to(\tau\to\rho))}.\lambda x^{\tau}.\lambda y^{\sigma}.f(x)(y)$$

This function basically rescopes the λ operators on variables in a λ-term. A different form of **perm** can be used for every permutation of the λ operators.

We can use **perm** to show that the various permutations of the arguments are type-logically equivalent.

Similarly, we can do a like operation for function composition:

$$\mathbf{comp}_{\sigma,\tau,\rho} \stackrel{\text{def}}{=} \lambda g^{\tau\to\rho}.\lambda f^{\sigma\to\tau}.\lambda x^{\sigma}.g(f(x))$$

Again, we can rescope terms freely using **comp**.

Note the following definitions:

- $\beta \circ \alpha \stackrel{\text{def}}{=} \mathbf{comp}(\beta)(\alpha)$

$$(\beta \circ \alpha)(\delta) \equiv (\mathbf{comp}(\beta)(\alpha))(\delta) \equiv \beta(\alpha(\delta))$$

- $\mathbf{I}_{\tau} \stackrel{\text{def}}{=} \lambda x^{\tau}.x$

$$\mathbf{I}(\alpha) \equiv (\lambda x.x)(\alpha) \equiv \alpha$$

A Proof Theory for the Simply Typed λ-calculus

The following are the Curry-Feys axioms, which can be treated like a simple rewriting system:

α-reduction:
$\vdash \lambda x.\alpha \Rightarrow \lambda y.(\alpha[x \mapsto y])$
$y \notin \text{Free}(\alpha)$ and y is free for x in α.

β-reduction:
$\vdash (\lambda x.\alpha)(\beta) \Rightarrow \alpha[x \mapsto \beta]$
β free for x in α.

η-reduction:
$\vdash \lambda x.(\alpha(x)) \Rightarrow \alpha$
$x \notin \text{Free}(\alpha)$

The rules of inference for the λ-calculus take the closure of the axioms. The rules allow not only for transitive and reflexive closure, but also for the *congruence closure*: logically equivalent terms bay be freely substituted one for the other, preserving truth.
Rules of inference: The rules of inference for the simply typed λ-calculus consist of all instances of the following schemata:

1. Reflexivity
   $\vdash \alpha \Rightarrow \alpha$
2. Transitivity
   $\alpha \Rightarrow \beta, \beta \Rightarrow \gamma \vdash \alpha \Rightarrow \gamma$
3. Congruence
   $\alpha \Rightarrow \alpha', \beta \Rightarrow \beta' \vdash \alpha(\beta) \Rightarrow \alpha'(\beta')$
4. Congruence
   $\alpha \Rightarrow \alpha' \vdash \lambda x.\alpha \Rightarrow \lambda x.\alpha'$
5. Equivalence
   $\alpha \Rightarrow \gamma, \beta \Rightarrow \gamma \vdash \alpha \Leftrightarrow \beta$

Proof: A proof of φ from $\mathbf{\Psi}$ is a sequence $\varphi_0, \ldots, \varphi_n$ such that $\varphi = \varphi_n$ and for every φ_i, one of the following holds:

a. $\varphi_i \in \mathbf{\Psi}$;

b. φ_i is an axiom;

c. $\mathbf{\Phi} \vdash \varphi_i$ is an inference rule with $\mathbf{\Phi} \subseteq \{\varphi_0, \ldots, \varphi_{i-1}\}$.

Lemma: If $x \notin \text{Free}(\alpha)$ then $[\![\alpha]\!]^{\theta}_{\mathcal{M}} = [\![\alpha]\!]^{\theta[x:=a]}_{\mathcal{M}}$

Sketch: The proof is a trivial induction on α. The triviality follows from the fact that $x \notin \text{Free}(\alpha)$.

Theorem: Soundness
If $\vdash \alpha \Rightarrow \beta$ then $\alpha \equiv \beta$.

Sketch: First the axioms must be verified. Having done that as a base, one does induction on terms. The proof requires a number of cases and, so, is rather tiresome.

Theorem: Church-Rosser
If $\vdash \alpha \Rightarrow \beta$ and $\vdash \alpha \Rightarrow \gamma$ then there is some δ such that $\vdash \beta \Rightarrow \delta$ and $\vdash \gamma \Rightarrow \delta$

Sketch: The proof uses induction on the structure of terms to show that if two terms can be rewrrten using an axiom applied to a subterm in one step, then there is a further sequence of steps that can reduce them to a common term.

Church-Rosser means, in essence, that the order in which subterms of a term are reduced is not important. The intermediate results can always be made equivalent via further rewritings.

Furthermore, reduction always terminates after a finite number of steps, resulting in a term that cannot be further reduced. Such a term is in normal form:

> *β, η Normal Form*
> A λ-term is in β, η normal form if and only if there is no subterm γ of α that has an alphabetic variant $\gamma' = \gamma$ such that γ' is eather a β-redex or an η-redex.

The following is a theorem, although the proof is distressingly complicated:

> *Strong Normalization*
> There are no infinitely long sequences of terms $\alpha_1, \ldots, \alpha_n, \ldots$ such that for $i > 0, \vdash \alpha_i \Rightarrow \alpha i + 1$ and $\alpha_i \neq_\alpha \alpha_{i+1}$.

Lemma: If α and β are in normal form, then $\alpha \equiv \beta$ if and only if $\alpha =_\alpha \beta$.

Sketch: Right to left is immediate. We must show that if $\alpha \neq_\alpha \beta$ then $\alpha \not\equiv \beta$. If they're of different types, the result is trivial. Otherwise, the proof proceeds by induction on the structure of α.

Theorem: Completeness
Two λ-terms α and β are logically equivalent only if $\vdash \alpha \Leftrightarrow \beta$ is provable.

Proof: Suppose α' and β' are in normal form such that $\vdash \alpha \Rightarrow \alpha'$ and $\vdash \beta \Rightarrow \beta'$. If $\alpha' =_\alpha \beta'$ then $\vdash \alpha \Rightarrow \beta'$ and $\vdash \alpha \Leftrightarrow \beta$.

Note that unlike the case of ordinary first order logic, the question of whether or not two λ-terms are logically equivalent is decidable:

> Theorem: Decidability
> There is an algorithm for deciding whether two λ-terms, α and β, are logically equivalent.

Proof: Normalize α and β. This process terminates after a finite number of steps. Check to see if the results are alphabetic variants. The latter is clearly a decidable relation.

Products

It's useful to think about how we might handle n-ary relations, keeping **BasTyp** constant. We would need to add a new clause to our definition of types, one which creates *product types*:

$$(\sigma \times \tau) \in \mathbf{Typ} \text{ if } \sigma, \tau \in \mathbf{Typ}$$

This allows us to create straightforward types for binary and ternary relations:

$$((\mathbf{Ind} \times \mathbf{Ind}) \to \mathbf{Bool})$$

$$(((\mathbf{Ind} \times \mathbf{Ind}) \times \mathbf{Ind}) \to \mathbf{Bool})$$

$$((\mathbf{Ind} \times (\mathbf{Ind} \times \mathbf{Ind})) \to \mathbf{Bool})$$

Note that the type for ternary relations is uninterestingly ambiguous. We, thus, want:

$$\langle a, b, c \rangle \stackrel{\text{def}}{=} \langle a, \langle b, c \rangle \rangle$$

Naturally, we require constants and variables for the new types:

a. $\mathbf{Con}_{\sigma\times\tau} \subseteq \mathbf{Term}_{\sigma\times\tau}$

b. $\mathbf{Var}_{\sigma\times\tau} \subseteq \mathbf{Term}_{\sigma\times\tau}$

Next, we need ways of referring to the subparts of product types:

a. $\langle\alpha, \beta\rangle \in \mathbf{Term}_{\sigma\times\tau}$ if $\alpha \in \mathbf{Term}_{\sigma}$ and $\beta \in \mathbf{Term}_{\tau}$

b. $\pi_1(\alpha) \in \mathbf{Term}_{\sigma}$ if $\alpha \in \mathbf{Term}_{\sigma\times\tau}$

c. $\pi_2(\alpha) \in \mathbf{Term}_{\tau}$ if $\alpha \in \mathbf{Term}_{\sigma\times\tau}$

π_1 and π_2 are *projection functions*.

Products must now be added to frames. To do this, we must define the domain of interpretation for product types:

$$\mathbf{Dom}_{\sigma\times\tau} = \mathbf{Dom}_{\sigma} \times \mathbf{Dom}_{\tau}$$

That is, a product type $(\sigma \times \tau)$ is interpreted in the domain of ordered pairs where the first element is in the domain of type σ and the second element is in the domain of type τ.

Models are still pairs consisting of a frame and an interpretation funtion $[\![\cdot]\!]^{\theta}_{\mathcal{M}}$:

1. $[\![c]\!] \in \mathbf{Dom}_{\sigma\times\tau}$ if $c \in \mathbf{Con}_{\sigma\times\tau}$;
2. $\theta(x) \in \mathbf{Dom}_{\sigma\times\tau}$ if $x \in \mathbf{Var}_{\sigma\times\tau}$
3. $[\![\langle\alpha, \beta]\!]^{\theta}_{\mathcal{M}} = \langle [\![\alpha]\!]^{\theta}_{\mathcal{M}}, [\![\beta]\!]^{\theta}_{\mathcal{M}}\rangle$
4. $[\![\pi_1(\alpha)]\!]^{\theta}_{\mathcal{M}} = a$ if $[\![\alpha]\!]^{\theta}_{\mathcal{M}} = \langle a, b\rangle$
5. $[\![\pi_2(\alpha)]\!]^{\theta}_{\mathcal{M}} = b$ if $[\![\alpha]\!]^{\theta}_{\mathcal{M}} = \langle a, b\rangle$

Finally, we can extend our proof system by means of the addition of a few simple axioms:

1. Left Projection
   $\vdash \pi_1(\langle \alpha, \beta \rangle) \Rightarrow \alpha$
2. Right Projection
   $\vdash \pi_2(\langle \alpha, \beta \rangle) \Rightarrow \beta$
3. Pairing
   $\vdash \langle \pi_1(\alpha), \pi_2(\alpha) \rangle \Rightarrow \alpha$
4. Pairing Congruence
   $\alpha \Rightarrow \alpha', \beta \Rightarrow \beta' \vdash \langle \alpha, \beta \rangle \Rightarrow \langle \alpha', \beta' \rangle$
5. Projection Congruence
   $\alpha \Rightarrow \alpha' \vdash \pi_1(\alpha) \Rightarrow \pi_1(\alpha')$
6. Projection Congruence
   $\alpha \Rightarrow \alpha' \vdash \pi_2(\alpha) \Rightarrow \pi_2(\alpha')$

Furthermore we can define the following combinators:

$$\mathbf{comm} \stackrel{\text{def}}{=} \lambda x.\langle \pi_2(x), \pi_1(x) \rangle$$

$$\mathbf{assoc} \stackrel{\text{def}}{=} \lambda x.\langle \langle \pi_1(x), \pi_1(\pi_2(x)) \rangle \pi_2(\pi_2(x)) \rangle$$

$$\mathbf{curry} \stackrel{\text{def}}{=} \lambda x^{(\sigma \times \tau) \to \rho}.\lambda y^{\sigma}.\lambda z^{\tau}.x(\langle y, z \rangle)$$

$$\mathbf{uncurry} \stackrel{\text{def}}{=} \lambda x^{(\sigma \to \tau) \to \rho}.y^{\sigma \times \tau}.x(\pi_1(y))(\pi_2(y))$$

The combinators **curry** and **uncurry** are particularly interesting since they show that one can go back and forth between a function defined on pairs and a function that takes the elements of a pair one at a time to produce a result. In particular, assuming everything is properly typed, we have the following:

$$\mathbf{curry}(\mathbf{uncurry}(\alpha)) \equiv \alpha$$

$$\mathbf{uncurry}(\mathbf{curry}(\beta)) \equiv \beta$$

Thus, we have a one-to-one relation between objects of type $(\sigma \times \tau) \to \rho$ and objects of type $(\sigma \to \tau) \to \rho$.