Variable-free Formalization of the Curry-Howard Theory

W. W. Tait

The reduction of the lambda calculus to the theory of combinators in [Schönfinkel, 1924] applies to *positive implicational logic*, i.e. to the typed lambda calculus, where the types are built up from atomic types by means of the operation $A \longrightarrow B$, to show that the lambda operator can be eliminated in favor of combinators K and S of each type $A \longrightarrow (B \longrightarrow A)$ and $(A \longrightarrow (B \longrightarrow C)) \longrightarrow ((A \longrightarrow B) \longrightarrow (A \longrightarrow C))$, respectively.[1] I will extend that result to the case in which the types are built up by means of the general function type $\forall x : A.B(x)$ as well as the disjoint union type $\exists x : A.B(x)$– essentially to the theory of [Howard, 1980]. To extend the treatment of $\longrightarrow$ to $\forall$ we shall need a generalized form of the combinators K and S, and to deal with $\exists$ we will need to introduce a new form of the combinator S (whose type turns out to be a general form of the Axiom of Choice). But also in the present context, if we are to eliminate variables, then not only the lambda operator for forming terms, but also quantification as a variable-binding operation for forming formulas must be analyzed away; so we will need an analogue of the combinators for formulas.

As usual, we shall write st for the value $s(t)$ of the function s for the argument t; so rst is $(r(s))(t)$, etc.

Let v be a free variable of type A. We wish to rewrite the formulas $B(v)$, $\forall x{:}A.B(x)$ and $\exists x{:}A.B(x)$, respectively, as $B'v$, $\forall B'$ and $\exists B'$, where B' is a type-valued function on A. If $t(v)$ is a term of type $B(v)$, which we express by

$$t(v) : B(v)$$

then $\lambda x{:}A.t(x)$ is a term of type $\forall x{:}A.B(x)$, denoting a function on A whose value for $s : A$ is $t(s) : B(s)$. We wish to rewrite the terms $t(v), \lambda x : A.t(x)$,

[1]This observation is essentially contained in the discussion of the so-called theory of functionality in Chapters 9 and 10 of [Curry and Feys, 1958].

respectively, as $t'v : B'v$ and $t' : \forall B'$. Thus, a two-quantifier formula

$$Q_1 x : A Q_2 y : B(x).C(x, y)$$

where Q_1 and Q_2 are quantifiers, is to be rewritten as

$$Q_1 x : A Q_2 y : B(x).C(x)'y$$

or

$$Q_1 x : A Q_2 y : B'x.C''xy$$

or simply

$$Q_1 Q_2 C''$$

C'' is a function defined on A such that $C''s$ is a type-valued function defined on $B's$ for all $s : A$. Let u and v be free variables of types A and B(u), respectively. A term $t(u, v)$ of type $C(u, v)$ should be rewritten as $t''uv$, where t'' is of type $\forall\forall C''$.

To discuss the general case, we need a definition.

Definition 1 *The notion of a* base of functionals *is defined by induction:*

- *The null sequence is a base.*
- *If A is a type and $F_1, \dots, F_n$ are functions defined on A such that, for each $t : A$, $\langle F_1 t, \dots, F_n t \rangle$ is a base, then the sequence $\langle A, F_1, \dots, F_n \rangle$ is a base.*

When $\langle A, F_1, \dots, F_n \rangle$ is a base, the base $\langle A, F_1, \dots, F_{n-1} \rangle$ is uniquely determined by the functional F_n. As an example, in the two-quantifier example above, $\langle A, B', C'' \rangle$ is a base. More generally, an n-quantifier formula

$$(1) \qquad Q_1 x_1 : A_1 \; Q_2 x_2 : A_2(x_1) \cdots Q_n x_n : A_n(x_1, \dots, x_{n-1}).B(x_1, \dots, x_n)$$

is to be rewritten as

$$Q_1 x_1 : A_1 \; Q_2 x_2 : A'_2 x_1 \cdots Q_n x_n : A_n^{(n-1)} x_1 \cdots x_{n-1}.B^{(n)} x_1 \cdots x_n$$

where $\langle A, A'_2, \dots A_n^{(n-1)}, B^{(n)} \rangle$ is a base, or simply as

$$Q_1 \cdots Q_n B^{(n)}.$$

If $v_1, v_2, \dots, v_n$ are free variables of types $A_1, A_2(v_1), \dots, A_n(v_1, \dots, v_n)$, respectively, then a term $t(v_1, v_2, \dots, v_n)$ of type $B(v_1, v_2, \dots, v_n)$ is to be rewritten as $t^{(n)} v_1 v_2 \cdots v_n$, where $t^{(n)}$ is of type $\forall\forall \cdots \forall B^{(n)}$.

In order to carry out this analysis, we need to introduce a formalism in which we can represent functionals and objects which depend upon free variables.

1 The Calculus

We must simultaneously define three notions:

- The notion of a *base of formulas*.
  - Bases are finite sequences whose members are called *formulas*.
  - If $\langle \vec{F}, G \rangle$ is a base, then $\vec{F}$ is called the *base* of G and denoted by $Base(G)$.
  - When $\langle A \rangle$ is a base, A is called a *(formula) type*.
  - A base of formulas is intended to denote a base of functionals for suitable values of the free variables.
  - With a formula we may associate a *rule of conversion*, which specifies the meaning of the formula. $FCONVG$ means that the formula F converts to the formula G according to the rules of conversion.
- The notion of a *term of type A*, where A is a type.
  - That t is a term of type A is expressed by $t\!:\!A$.
  - With a term we may associate a *rule of conversion*, which specifies the meaning of the term. $sCONVt$ means that the term s converts to the term t according to the rules of conversion.
- The notion of *definitional equality* between two terms or between two functionals.
  - We denote this relation by $\equiv$.
  - We may specify at once that, for terms s and t, $s \equiv t$ is defined to mean $s\ RED\ r \wedge t\ RED\ r$ for some r, where the relation RED is defined in terms of the rules of conversion: call an occurrence of a formula or term X in a formula or term U *external* if it is not in a part of U of the form $v_n(A)$. (When A is a formula, $v_n(A)$ will be introduced as a variable of type A.) For formulas or terms U and V, $U > V$ will mean that V is obtained by replacing some external occurrence X of U by Y, where $X\ CONV\ Y$. RED is the least reflexive and transitive relation which includes $>$.

– For formulas F and G, $F \equiv G$ will mean that the base of F and the base of G are of the same length $n \geq 0$ and, for some distinct new symbols $x_1, \ldots, x_n$, $Fx_1 \cdots x_n$ and $Gx_1 \cdots x_n$ RED to a common expression.[2]

– We may also specify at once that the type of a term is to be determined only to within definitional equality. Thus, as a part of the definition of the type relation we specify that

$$t{:}A \wedge A \equiv B \longrightarrow t{:}B.$$

It will follow that

$$s \equiv t \wedge s{:}A \longrightarrow t{:}A.$$

If $\vec{Y} = \langle Y_1, \ldots, Y_n \rangle$, then $\langle X, \vec{Y} \rangle$ will denote $\langle X, Y_1, \ldots, Y_n \rangle$, $\langle \vec{Y}, Z \rangle$ will denote $\langle Y_1, \ldots, Y_n, Z \rangle$, $\vec{Y}t$ will denote $\langle Y_1 t, \ldots, Y_n t \rangle$, etc.

1.1 Atomic Formulas

If $\vec{F}$ is a base of formulas none of which contains free variables, then $R_n(\vec{F})$ is an atomic formula with base $\vec{F}$ for each n. There may be conversion rules associated with an atomic formula.

1.2 Instantiation

If G has base $\langle A, \vec{F} \rangle$ and $t{:}A$, then Gt is a formula with base $\vec{F}t$.

1.3 Quantification

If H has base $\langle \vec{F}, G \rangle$, then $\forall H$ and $\exists H$ are formulas with base $\vec{F}$.

- If $\vec{F}$ is not null and Q is a quantifier, then we have the conversion rule

$$(QH)t \ CONV \ Q(Ht)$$

[2]Notice that, on our definition, variables $v_n(A)$ are always in *normal form*, where a formula or term X is in normal form iff there is no Y such that $X > Y$. Thus, even when the distinct types A and B are $\equiv$, $v_n(A) \not\equiv v_n(B)$.

- The *(universal) closure* of a formula H is

$$H^* = \forall \cdots \forall H$$

  where the number of $\forall$'s is the length of the base of H. Thus, H^* is a type.

1.4 Dummy Argument Places

If $\langle \vec{F}, G \rangle$ and $\langle \vec{F}, H_1, \ldots, H_k \rangle$ are bases, then so is $\langle \vec{F}, G, H_1[G], \ldots, H_k[G] \rangle$.

- The rules of conversion for $H_i[G]$ $(i = 1. \ldots, k)$ are:
  - If $\vec{F} \neq \emptyset$

$$H_i[G]t \; CONV \; H_it[Gt]$$

  - If $\vec{F} = \emptyset$

$$H_i[G]t \; CONV \; H$$

- Abbreviations: Let $Base(G) = Base(H)$

$$G \longrightarrow H = \forall(H[G])$$

$$G \wedge H = \exists(H[G])$$

1.5 Transposition of Argument Places

If $\langle \vec{E}, F, G, H_1, \ldots, H_k \rangle$ is a base, then so is $\langle \vec{E}, \forall G, F[\forall G], H_1\{1\}, \ldots, H_k\{k\} \rangle$.

The subscript 'i' in H_i is meta-notation, marking which formula in the base we are refering to; the '$\{i\}$', on the other hand, is part of the syntax of the formula $H_i\{i\}$. The rules of conversion are:

- If $\vec{E} \neq \emptyset$

$$H_i\{i\}t \; CONV \; H_it\{i\}$$

- If $\vec{E} = \emptyset$

$$H_i\{i\}st \; CONV \; H_it(st)$$

REMARK. In the second case, note that s must be a term of type $\forall G$ and t must be of type $F[\forall G]s$, i.e. of type F. Since G has base F, st is defined and is of type Gt, by the principle of $\forall$ Elimination in §1.8 below. So $H_it(st)$ is defined.

1.6 Variables

For each type A and $n \geq 0$

$$v_n(A) : A$$

$v_n(A)$ is called a *free variable* of basic type A. Note that A is a syntactical part of $v_n(A)$. A variable of basic type A may be denoted by $v(A)$, $v'(A)$, etc.

1.7 Constants

If A is a type containing no variables, zero or more constant terms of type A may be introduced.

1.8 Quantifier Elimination

Let $\langle A, F \rangle$ be a base.

- $\forall$ Elimination

$$s : A,\ t : \forall F \Longrightarrow ts : Fs$$

- $\exists$ Elimination

$$p : \exists F \Longrightarrow (p1) : A,\ (p2) : F(p1)$$

1.9 Existential Quantifier Introduction

Let H have base $\langle \vec{F}, G \rangle$.

$$P(H) : (H \longrightarrow ((\exists H)[G])^*$$

The conversion rules for $\exists$ are

- If $\vec{F} \neq \emptyset$

$$P(H)t\ CONV\ P(Ht)$$

- If $\vec{F} = \emptyset$

$$P(H)st1\ CONV\ s$$

$$P(H)st2\ CONV\ t$$

1.10 The Combinator K

Let G and H have base $\vec{F}$.

$$K(G,H):(G \longrightarrow (H \longrightarrow G))^*$$

The conversion rules associated with K are

- If $\vec{F} \neq \emptyset$
$$K(G,H)t \ CONV \ K(Gt,Ht)$$
- If $\vec{F} = \emptyset$
$$K(G,H)st \ CONV \ s$$

1.11 The Combinators $S_\forall$ and $S_\exists$

Let H have base $\langle \vec{E}, F, G\rangle$ and let Q be a quantifier $\forall$ or $\exists$. Then

$$S_Q(H):(\forall QH \longrightarrow Q\forall(H\{1\})^*$$

The conversion rules are

- If $\vec{E} \neq \emptyset$
$$S_Q(H)t \ CONV \ S_Q(Ht)$$
- Assume that $\vec{E} = \emptyset$ and let $r:\forall QH$. So $H\{1\}$ has base $\langle \forall G, F[\forall G]\rangle$. Let $s:\forall G$ and $t:F[\forall G]s$. So $t:F$ and
$$S_Q(H)r:\forall Q(H\{1\})$$
  - Let $Q = \forall$.
$$S_\forall(H)r:\forall\forall(H\{1\})$$
    $S_\forall(H)rst$ must be defined to be of type $H\{1\}st$, i.e. of type $Ht(st)$. But $rt:\forall(Ht)$, $st:Gt$ and so $rt(st):Ht(st)$. Thus, we may define $S_\forall(H)rst$ by the conversion rule
$$S_\forall(H)rst \ CONV \ rt(st)$$

– Let $Q = \exists$.

$$S_\exists(H)r : \exists\forall(H\{1\}$$

Thus

$$S_\exists(H)r1 : \forall G$$

$$S_\exists(H)r2 : \forall H\{1\}(S_\exists(H)r1)$$

So

$$S_\exists(H)r1t : Gt$$

$$S_\exists(H)r2t : Ht(S_\exists(H)r1t)$$

But $rt : \exists Ht$ and so $rt1 : Gt$ and $rt2 : Ht(rt1)$. So we may define $S_\exists(H)$ by the conversion rules

$$S_\exists(H)r1t\ CONV\ rt1$$

$$S_\exists(H)r2t\ CONV\ rt2$$

We have completed the description of the calculus.

Notice that the type of $S_\exists(H)$ is a general form of the Axiom of Choice: for example, let H have base $\langle A, B[A]\rangle$. Then $H\{1\}$ has base $\langle A \longrightarrow B, A[A \longrightarrow B]\rangle$ and the type $\forall\exists H \longrightarrow \exists\forall H\{1\}$ may be written as

$$\forall x : A \exists y : B Hxy \longrightarrow \exists f : A \longrightarrow B \forall x : A Hx(fx)$$

2 Some properties of the calculus

Let $Var(X)$ denote the set of variables in the formula or term X.

Definition 2 *The type B of the term t is* suitable *for t iff $Var(B) = Var(t) - \{t\}$.*

Lemma *The following facts are easily derived.*

1. *Every variable in a formula in the base of F is in F.*
2. *Every term has a suitable type.*
3. *If G and H have bases $\vec{E}$ and $\langle \vec{E}, F\rangle$, respectively, then*

$$\forall(H[G]) \equiv (\forall H)[G]$$

.

4. If G and H both have base $\langle \vec{E}, F\rangle$, then

$$H[G]\{1\} \equiv H[\forall G]$$

5. Let F, G and H all have base $\vec{E}$. Then

$$H[F][G[F]] \equiv H[G][F]$$

.

Assuming that there are no further conversion rules, we may prove in the usual way

Theorem 1 *Church-Rosser Theorem If the formula or term X reduces to Y and to Z, then Y and Z reduce to some U. In particular, every term or formula has at most one normal form.*

Theorem 2 *Well-foundedness Theorem If X is a formula or term, then every sequence $X > Y > \cdots$ is finite. In particular, every formula or term has a normal form.*

In view of these two theorems, the relation $\equiv$ between formulas and terms is decidable. We will not discuss general conditions on extensions of the calculus obtained by adding new conversion rules under which the Church-Rosser and Well-foundedness Theorems are preserved, since the main result of this paper, the Explicit Definition Theorem below, will be preserved by any such extension.

3 Identity Function

Let G and H have base $\vec{F}$ and let $S = S_\forall(H[G])$. Then S is of type

$$(\forall(G \longrightarrow H) \longrightarrow \forall\forall(H[G]\{1\}))^*$$

which, by 3 and 4 of the Lemma is $\equiv$ to

$$(\forall(G \longrightarrow H) \longrightarrow (\forall G \longrightarrow \forall H))^* \qquad (2)$$

Let G be $B[A]$ and let H be $C[A]$. By 5 of the Lemma, (2) is $\equiv$ to

$$(A \longrightarrow (B \longrightarrow C)) \longrightarrow ((A \longrightarrow B) \longrightarrow (A \longrightarrow C)) \tag{3}$$

So

$$S{:}(A \longrightarrow (B \longrightarrow C)) \longrightarrow ((A \longrightarrow B) \longrightarrow (A \longrightarrow C))$$

Set $B = A \longrightarrow A$, $C = A$, $K_1 = K(A, B)$ and $K_2 = K(A, A)$. Then $K_1{:}A \longrightarrow (B \longrightarrow C)$ and $K_2{:}A \longrightarrow B$. Set

$$I_A = SK_1K_2$$

Then $I_A{:}A \longrightarrow C$, i.e. $I_A{:}A \longrightarrow A$. Let $t{:}A$.

$$I_At = SK_1K_2t \equiv K_1t(K_2t) \equiv t.$$

Thus I_A is the identity function on A.

Notice that the combinators for positive implicational logic really are a special case of $K(G, H)$ and $S_\forall(H)$. Namely, they are $K(A, B)$ of type $A \longrightarrow (B \longrightarrow A)$ and $S_\forall(C[A][B[A]])$ of type (3).

4 Explicit Definition Theorem

Definition 3 *A variable v is* unfettered *in the term t (formula F) iff for every variable $v(A)$ occuring in t (F), v does not occur in A.*

Note: If B is a suitable type for the term t, then v is unfettered in t iff it is unfettered in B.

Theorem *Explicit Definition Theorem Let $v = v(A)$.*

- *If $\langle F_1, \ldots, F_n\rangle$ is a base and v is unfettered in F_n, then there is a base $\langle A, F'_1, \ldots, F'_n\rangle$ such that $Var(F'_i) \subseteq Var(F_i) - \{v\}$ and*

$$F'_iv \ RED \ F_i$$

- *If $t : B$ and v is unfettered in t and in B, then there is a $t' : \forall B'$ such that $Var(t') \subseteq Var(t) - \{v\}$ and*

$$t'v \ RED \ t$$

Note: If $B \equiv C$, then $B' \equiv C'$. So, in particular, given a term t in which v is unfettered, we need only find one type C of t in which v is unfettered and construct $t' : \forall C'$. If B is another type of t in which v is unfettered, then t' will be of type $\forall B'$ as well.

Proof. The proof is by induction on the definition of the base or term.
Case 1. Assume that v does not occur in F_n. Then it does not occur in any F_i. Set $F'_i = F_i[A]$.
Case 2. Assume that v is not in t and let B be a suitable type for t. Then v is not in B and so $B' = B[A]$. Set $t' = K(B, A)t$, which is of type $\forall B' = A \longrightarrow B$ and $t'v$ $CONV$ t.

In the remaining cases, we may assume that v occurs in the formula or term in question.
Case 3. Let us assume that F' is defined for $F = G$, $F = H$ and for every formula F in the base of G or H. Then we may clearly set

$$(QH)' = QH'$$

$$H[G]' = H'[G']$$

$$H\{n\}' = H'\{n\}$$

$$P(H)' = P(H')$$

$$K(G, H)' = K(G', H')$$

$$S_Q(H)' = S_Q(H')$$

For example, $H[G]'v = H'[G']v$ $CONV$ $H'v[G'v]$ RED $H[G]$. And $K(G, H)'v = K(G', H')v$ $CONV$ $K(G'v, H'v)$ RED $K(G, H)$. Note that $K(G, H)'$ is of type $(G' \longrightarrow (H' \longrightarrow G')^*$, which is $\forall(G \longrightarrow (H \longrightarrow G)^*)'$, so the type is right.
Case 4. Let $F_i = G_i t$, where $t : C$ and $\langle C, G_1, \ldots, G_n\rangle$ is a base. Then $\langle A, C', G'_1, \ldots, G'_n\rangle$ is a base and $t' : \forall C'$. Set $F'_i = G'_i\{i\}t'$. Then

$$F'_i v \; CONV \; G'_i v(t'v) \; RED \; G_i t = F_i$$

Case 5. Let H have base $\langle B\rangle$, $f : \forall H$, and $t : B$. We need to define $(ft)'$. $f' : \forall\forall\ H'$, $t' : \forall B'$ and $S_\forall(H')f' : \forall\forall H'\{1\}$. $H'\{1\}$ has base $\langle \forall B', A[\forall B'\rangle$. So $S_\forall(H')f't'$ is defined and is of type $\forall H'\{1\}t' \equiv \forall(Ht)'$. So set $(ft)' = S_\forall(H')f't'$. For

$$S_\forall(H')f't'v \; RED \; f'v(t'v) \; RED \; ft$$

Case 6. Let $p\!:\!\exists H$, where H has base B. We need to define $(p1)'$ and $(p2)'$. $p' : \forall\exists H'$, where H' has base $\langle A, B'\rangle$. $H'\{1\}$ has base $\langle \forall B', A[\forall B']\rangle$. So $S_\exists(H')p'\!:\!\exists\forall(H'\{1\})$. Set $(p1)' = S_\exists(H')p'1$ and $(p2)' = S_\exists(H')p'2$.

$$(p1)'v\ RED\ p'v1\ RED p1$$

$$(p2)'v\ RED\ p'v2\ RED p2.$$

The proof is completed.

We may now take $\forall x : A.B(x)$ to be an abbreviation for $\forall B'$, providing the free variable $v = v(A)$ is unfettered in $B(v)$. If v is fettered in B, then B has the form $B(v, u(C(v)))$, where $u(C(v))$ is a variable and v is unfettered in $C(v)$. But in this case, $\forall x\!:\!A.B(x)$, i.e. $\forall x\!:\!A.B(x, u(C(x)))$ doesn't make any literal sense: $u(C(x))$ does not denote a variable of any particular type. Rather we can only think of it as a dependent variable, depending on the value of x. But then we may more accurately replace $u(C(v))$ by $u(\forall C')v$, eliminating at least one context which fetters v. Iterating this proceedure, we finally transform $B(v)$ into a type $D(v)$ in which v is unfettered and such that $\forall x\!:\!A.D(x)$ expresses the only reasonable meaning of $\forall x\!:\!A.B(x)$. Similarly, we may restrict $\lambda x\!:\!A.t(x)$ to the case in which v is unfettered in $t(v)$; and in that case it is an abbreviation for t'. In this case, the restriction that v be unfettered in $t(v)$ is precisely Gentzen's restriction on his rule $\forall - I$ in the system of natural deduction.

The second part of the theorem contains the *Deduction Theorem*, i.e. Gentzen's rule $\supset -I$: If v does not occur in B and is unfettered in the term t of type B' i.e. t is a deduction of B from the assumption v of A, then t' is a deduction of $A \longrightarrow B$ which depends only upon assumptions in t other than v.

Now we return to the initial discussion of the n-quantifier form. Let $B = B(v_1, \ldots, v_n)$ be a formula and $v_1, \ldots, v_n$ a list of variables including all the variables in B, $v_i = v_i(A_i)$. Assume that the list of variables is in *good order*, i.e. $i < j$ implies that v_j does not occur in A_i. So we may write $A_i = A_i(v_1, \ldots, v_{i-1})$, displaying all the free variables. Then v_n is unfettered in B and we may apply the Explicit Definition Theorem to obtain B' with base $\langle A_n\rangle$, containing at most the variables v_i for $i < n$ and such that $B'v_n \equiv B$. v_{n-1} is unfettered in B' and so we may construct B'' with base $\langle A_{n-1}, A_n'\rangle$, containing at most the variables v_i for $i < n-1$, such that $B''v_{n-1}v_n \equiv B$. Iterating n times, we obtain the variable-free formula

$B^{(n)}$ with base $\langle A_1, A'_2, \ldots, A_n^{(n-1)}\rangle$ such that $B^{(n)}v_1 \cdots v_n \equiv B$. Then (1) is precisely $Q_1 \cdots Q_n.B^{(n)}$. We denote B^n by

$$\lambda x_1 : A_1 \cdots \lambda_n x_n : A_n(x_1, \ldots, x_{n-1}).B(x_1, \ldots x_n).$$

Moreover, if $t = t(v_1, \ldots, v_n)$ is a term of type $B(v_1, \ldots, v_n)$, then n applications of the Explicit Definition Theorem yields $t^{(n)} : \forall \cdots \forall B^{(n)}$ with $t^{(n)}v_1 \cdots v_n \equiv t$. We denote t^n by

$$\lambda x_1 : A_1 \cdots \lambda_n x_n : A_n(x_1, \ldots, x_{n-1}).t(x_1, \ldots x_n).$$

For future reference, when $\vec{F} = \langle F_1, \ldots, F_n\rangle$ is a base and G a formula, we write

$$\lambda \vec{x} : \vec{F} = \lambda x_1 : F_1 \cdots \lambda x_n : F_n x_1 \ldots x_{n-1}.$$

and

$$G[\vec{F}] = \lambda \vec{x} : \vec{F}.G.$$

5 Truth Functions

We briefly discuss one extension of the basic formalism, obtained by adding the *null type* $NULL$, the *two-object type* TWO and the atomic formula $V = R_0(TWO)$ with base $\langle TWO\rangle$. The Church-Rosser and Well-foundedness Theorems hold for this extension.

There is no introduction rule for $NULL$. Let G have base $\vec{F}$.
$NULL$ Elimination

$$N(G) : (NULL[\vec{F}] \longrightarrow G)^*.$$

When $\vec{F}$ is not empty, there is the conversion rule

$$N(G)t \; CONV \; N(Gt)$$

Negation can now be defined by

$$\neg G = (G \longrightarrow NULL).$$

Concerning the Two-object Type,

- *TWO* Introduction

$$\top{:}TWO \qquad \bot{:}TWO$$

- *TWO* Elimination. Let G have base $(\vec{F}, TWO[\vec{F}])$

$$J(G){:}(\lambda\vec{x}{:}\vec{F}.G\vec{x}\top \longrightarrow (\lambda\vec{x}{:}\vec{F}.g\vec{x}\bot \longrightarrow \forall G))^*$$

The conversion rules are:

- When $\vec{F}$ is nonempty, the conversion rule is

$$J(G)r\ CONV\ J(Gr)$$

- When $\vec{F}$ is null, the conversion rules are

$$J(G)st\top\ CONV\ s \qquad J(G)st\bot\ CONV\ t$$

There are no introduction or elimination rules associated with V; there are just the conversions

$$V\top\ CONV\ (NULL \longrightarrow NULL) \qquad V\bot\ CONV\ NULL$$

Thus $V\top$ is the true proposition proved by $N(NULL)$ and $V\bot$ is a false proposition. Given formulas G and H with base $\vec{F}$, we define

$$\langle G, H\rangle = (V[\vec{F}] \longrightarrow G[TWO[\vec{F}]]) \wedge (\neg V \longrightarrow B[TWO[\vec{F}]])$$

with base $\langle \vec{F}, TWO[\vec{F}]\rangle$ When A and B are types, it is easy to construct proofs of

$$\langle A, B\rangle\top \longleftrightarrow A \qquad \langle A, B\rangle\bot \longleftrightarrow B$$

where by a *proof* of a type C we mean a term of type C which contain no variables not in C. In view of this, we may introduce disjunction of two formulas G and H with the same base by

$$G \vee H = \exists\langle G, H\rangle$$

Similarly we could define a new but equivalent conjunction by $\forall\langle G, H\rangle$.

References

Curry, H. and Feys, R. [1958]. *Combinatory Logic I*, Amsterdam: North-Holland.

Howard, W. [1980]. The formula-as-types notion of construction, *in* J. Hindley and J. Seldin (eds), *To H.B. Curry*, Academic Press, New York, pp. 479–490.

Schönfinkel, M. [1924]. Über die Bausteine der mathematische Logik, *Mathematische Annalen* **23**: 123–153.