

## II. Higher-Order Simple Type Theories (HOSTs)

### A. Syntax:

**Definition:** A *type symbol* is defined recursively as follows: (i)  $o$  is a type symbol, (ii) if  $\eta_1, \dots, \eta_n$  are type symbols, then  $(\eta_1 \dots, \eta_n)$  is a type symbol, and (iii) nothing that cannot be constructed from repeated applications of (i) and (ii) is a type symbol.

Examples (with intended meaning)

$o$  is the type of *individuals*

$(o)$  is the type of *properties* applicable to individuals.

$(o, o)$  is the type of *dyadic relations* applicable to individuals.

$((o))$  is the type of *properties* applicable to *properties* of type  $(o)$ .

$(o, (o))$  is the type-symbol of *dyadic relations* holding between *individuals* and a *properties of individuals*.

The use of the words “properties” and “relation” here may be controversial. It may be best to stick to the linguistic level: The type-symbol “ $o$ ” is the symbol for *individual terms*; the type-symbol  $(o)$  is the type for *monadic predicates* applied to individual terms, etc.

**Definition:** A *variable* is any lower or uppercase letter between  $f, \dots, z$ , written with or without a numerical subscript, and with a *type-symbol* as superscript.

Examples: “ $f^{(o)}$ ”, “ $x_2^{(o)}$ ”, “ $R_1^{(o, o, o)}$ ”.

**Definition:** A *constant* is any lower or uppercase letter between  $a, \dots, e$ , written with or without a numerical subscript, and with a *type-symbol* as superscript.

Different higher-order languages may use different sets of constants, or different subsets of the type-symbols.

**Definition:** A *well-formed expression (wfe)* is defined recursively as follows:

(i) a variable or constant is a wfe expression of the type given by its superscript;

(ii) if  $\phi$  is a wfe of type  $(\eta_1 \dots, \eta_n)$ , and  $\tau_1, \dots, \tau_n$  are wfes of types  $\eta_1, \dots, \eta_n$  respectively, then  $\phi(\tau_1, \dots, \tau_n)$  is a type-less wfe;

(iii) if  $\alpha$  and  $\beta$  are type-less wfes, then  $\alpha \vee \beta$  is a type-less wfe;

(iv) if  $\alpha$  is a type-less wfe, then  $\neg \alpha$  is a type-less wfe;

(v) if  $\alpha$  is a type-less wfe, and  $\xi$  is a variable, then  $((\forall \xi)\alpha)$  is a type-less wfe;

(vi) if  $\xi_1, \dots, \xi_n$  are distinct variables of types  $\eta_1, \dots, \eta_n$  respectively and  $\alpha$  is a type-less wfe, then  $[\lambda \xi_1 \dots \xi_n \alpha]$  is a wfe of type  $(\eta_1 \dots, \eta_n)$  (and all occurrences of those variables are considered bound in that wfe);

(vii) nothing that cannot be constructed by repeated applications of (i)-(vi) is a wfe.

**Definitions:** A type-less wfe is called a *well-formed formula (wff)*. A wfe that is not a wff is called a *term*.

Notice that although “ $\lambda$ ” is a Greek letter, we are here using it as part of our object language(s), and not as a metalinguistic variable.

Terms of the form  $[\lambda x \dots x \dots]$  might be read “the property of being an  $x$  such that  $\dots x \dots$ ”. Terms of the form  $[\lambda x y \dots x \dots y \dots]$  might be read “the relation that holds between  $x$  and  $y$  when  $\dots x \dots y \dots$ ”, although again, this phrasing may be controversial.

I use the notation  $[\lambda x \dots x \dots]$  where Hatcher would instead write  $\dots \hat{x} \dots$ . Hatcher’s notation is older, and now somewhat outdated, though the two notations are historically connected.

$\wedge, \rightarrow, \leftrightarrow, (\exists \xi)$  are defined as one would expect.

Typically,  $\tau = \mu$  is defined as  $(\forall \xi)(\xi(\tau) \leftrightarrow \xi(\mu))$ , where  $\tau$  and  $\mu$  have the same type,  $\eta$ , and  $\xi$  is the first variable of type  $(\eta)$  not occurring free in either  $\tau$  or  $\mu$ .

### B. Formulation

Different HOSTs have different axioms. However, all standard theories have as axioms (or theorems) all truth-table tautologies, and the following schemata:

- $(\forall \xi)\alpha(\xi) \rightarrow \alpha(\tau)$ , where  $\xi$  is a variable of any type, and  $\tau$  is any term of the same type as  $\xi$ , and no free variables of  $\tau$  become bound in  $\alpha(\tau)$ .
- $(\forall \xi)(\beta \rightarrow \alpha(\xi)) \rightarrow (\beta \rightarrow (\forall \xi)\alpha(\xi))$ , where  $\xi$  is a variable of any type, and provided that  $\beta$  does not contain  $\xi$  free.
- $(\forall \zeta_1) \dots (\forall \zeta_n)([\lambda \xi_1 \dots \xi_n \alpha(\xi_1, \dots, \xi_n)](\zeta_1, \dots, \zeta_n) \leftrightarrow \alpha(\zeta_1, \dots, \zeta_n))$ , where  $\xi_1, \dots, \xi_n, \zeta_1, \dots, \zeta_n$  are distinct variables and each  $\xi_i$  matches  $\zeta_i$  in type.

The inference rules are MP and UG (applicable to any type variable).

So in addition to the normal first-order instances of the above, we also have, e.g.:

$$\begin{aligned} & (\forall F^{(o)})(\exists x^{(o)}F^{(o)}(x^o) \rightarrow (\exists x^{(o)})[\lambda y^o y^o = y^o](x^o)) \\ & (\forall R^{(o,o)})(F^{(o)}(x^o) \rightarrow R^{(o,o)}(x^o, x^o)) \rightarrow \\ & \quad (F^{(o)}(x^o) \rightarrow (\forall R^{(o,o)})R^{(o,o)}(x^o, x^o)) \\ & (\forall G^{(o)})([\lambda F^{(o)} F^{(o)}(x^o)](G^{(o)}) \leftrightarrow G^{(o)}(x^o)) \end{aligned}$$

**Derived rules:** *lambda conversion* ( $\lambda$ -cnv): where no free variable of  $\tau_1, \dots, \tau_n$  becomes bound in the context  $\alpha(\tau_1, \dots, \tau_n)$ :

$$\begin{aligned} & [\lambda \xi_1 \dots \xi_n \alpha(\xi_1, \dots, \xi_n)](\tau_1, \dots, \tau_n) \vdash \alpha(\tau_1, \dots, \tau_n) \\ & \alpha(\tau_1, \dots, \tau_n) \vdash [\lambda \xi_1 \dots \xi_n \alpha(\xi_1, \dots, \xi_n)](\tau_1, \dots, \tau_n) \end{aligned}$$

**Definition:** The *pure higher-order predicate calculus* (HOPC) is the HOST whose *only* axioms are instances of the above schemata in the higher-order language that contains *no* constants.

Begin counting at 0, and parse a type-symbol, adding 1 for every ‘(’ and subtracting one for every ‘)’. The highest number reached while parsing is the *order* of the type-symbol.

**Definition:** A theory of order  $n$  is just like HOST, except eliminating from the syntax all variables of order  $n$  or greater and all constants of order  $n+1$  or greater.

**Abbreviation convention:** Type-symbol superscripts may be left off variables in later occurrences in the same wff, provided that the same letter and subscript are not also used in that wff for a variable of a different type.

Many HOSTs have the following as axioms or theorems, including the versions of type-theory outlined by Hatcher:

*The axiom (principle) of extensionality (Ext):*

$$\begin{aligned} & (\forall F^{(\eta_1, \dots, \eta_n)})(\forall G^{(\eta_1, \dots, \eta_n)})(\forall x_1^{\eta_1} \dots \forall x_n^{\eta_n}) \\ & \quad (F(x_1^{\eta_1}, \dots, x_n^{\eta_n}) \leftrightarrow G(x_1^{\eta_1}, \dots, x_n^{\eta_n})) \rightarrow F = G \end{aligned}$$

*The axiom (principle) of infinity (Inf):*

$$\begin{aligned} & (\exists R^{(o,o)})(\forall x^o \neg R(x, x) \wedge (\forall x)(\exists y^o)R(x, y) \wedge \\ & \quad (\forall x)(\forall y)(\forall z^o)(R(x, y) \wedge R(y, z) \rightarrow R(x, z))) \end{aligned}$$

(Ext) forces us to think of the values of the higher-order variables as entities with *extensional* identity conditions, such as sets or classes. We may also think of the them as Fregean functions from objects to truth-values, or something similar. However, we may not think of them as Platonic universals or other intensionally-individuated entities. Thus we have as the simplest instance:

$$(\forall F^{(o)})(\forall G^{(o)})(\forall x^o)(Fx \leftrightarrow Gx) \rightarrow F = G$$

This says that “properties” of individuals are identical when they apply to all and only the same individuals.

With (Ext), we are free to regard the values of the variables of type  $(o)$  simply as *classes of individuals*, and the variables of type  $((o))$  as *classes of classes*, etc. We are then free to regard the notation  $[\lambda \xi \alpha(\xi)]$  as a variant of  $\{\xi \mid \alpha(\xi)\}$  and the notation  $\xi(\zeta)$  as simply a variant of  $\zeta \in \xi$ .

With both (Ext) and (Inf) added to (HOPC) we can develop Peano arithmetic *precisely* as we did for ST. The principle of Lambda conversion is a more general version of ST1, (Ext) is a more general version of ST2, and (Inf) replaces ST3.

(HOPC + (Ext) + (Inf) is, in effect, Hatcher’s TT.)

*Homework:* (a) Prove the following:

$$\begin{aligned} & (a) \vdash_{\text{HOPC}} (\forall x^n)(x^n = x^n) \\ & (b) \vdash_{\text{HOPC}} (\forall x^n)(\forall y^n)(x^n = y^n \rightarrow (\alpha(x^n, x^n) \rightarrow \alpha(x^n, y^n))) \\ & \quad \text{where } \alpha(x^n, x^n) \text{ does contain } y^n \text{ bound.} \end{aligned}$$

(c) Explain why it is unnecessary to take the quantifier  $(\forall \xi)$  as a primitive variable binding operator in the syntax of HOSTs; one may instead use *constants* of the form  $A^{(n)}$ .

### C. Doing without (Ext) or (Inf)

Is it possible to define numbers and capture arithmetic without assuming (Ext)?

Russell used a version of type-theory in which class-abstracts we introduced via contextual definitions. The general approach could be summarized as follows:

The wff  $\beta(\{\xi \mid \alpha(\xi)\})$  is an abbreviation for the wff  $(\exists \phi)((\forall \xi)(\phi(\xi) \leftrightarrow \alpha(\xi)) \wedge \beta(\phi))$ , where  $\phi$  is the first (predicative) variable of type  $(\eta)$ , where  $\eta$  is the type of the variable  $\xi$ ,

This approach was of a piece with his theory of descriptions, the view that a wff of the form  $\beta(\{\xi \mid \alpha(\xi)\})$  is to be regarded as an abbreviation of a wff of the form  $(\exists \xi)((\forall \zeta)(\alpha(\zeta) \leftrightarrow \xi = \zeta) \wedge \beta(\xi))$ .

Both kinds of contextual definition give rise to scope ambiguities:

$$\begin{aligned} & \neg M^{(o)}(\{x^o \mid x^o = x^o\}) \text{ may either mean} \\ & (\exists F^{(o)})(\forall x^o)(F(x) \leftrightarrow x = x) \wedge \neg M^{(o)}(F^{(o)}), \text{ or} \\ & \neg (\exists F^{(o)})(\forall x^o)(F(x) \leftrightarrow x = x) \wedge M^{(o)}(F^{(o)}) \end{aligned}$$