Stanford Encyclopedia of Philosophy

**Supplement to Alonzo Church**

# D. The λ-Calculus and Type Theory



---

## D.1 Church's Lambda Calculus

According to Church, a

> function is a rule of correspondence by which when anything is given (as argument) another thing (the value of the function for that argument) may be obtained. (1941 [BE: 201])

The λ-calculi are essentially a family of notations for representing functions as such rules of correspondence rather than as graphs (i.e., sets or classes of ordered pairs of arguments and values).

The λ-operator is a monadic variable binding "abstraction" operator that allows one to build complex expressions that designate functions. For example, the occurrence of the numerical successor function—henceforth "$S(x)$"—in the form "$S(x) > 1000$" is meaningless until we pick a particular natural number as the value for "$x$". In contrast, the occurrence of "$S(x)$" in the form "$S(x)$ is primitive recursive" *designates a particular function*. Using "λ" we can build an expression that designates this function by writing

$$\lambda x[S(x)].$$

We can also introduce *n*-adic λ-abstracts that designate *n*-ary functions (or functions of $n$ variables) such as the dyadic

$$\lambda xy[x + y].$$

Complex predicates including those from natural language can also be treated as functions—from individuals to truth values. For this reason, λ-abstracts have become part of the meta-language of formal semantics.(see Partee et al. 1990 as well as Heim & Kratzer 1998—two enormously influential textbooks). For example, from the form "$x$ is a Polish diplomat and $x$ is a great pianist", we can obtain the corresponding complex predicate:

$$\lambda x[x \text{ is a Polish diplomat and } x \text{ is a great pianist}].$$

Intuitively, this can be read as "is an individual who is both a Polish diplomat and a great pianist". One can also build compound *n*-adic λ-abstracts from compound formulas. For example, from the form "$x$ is evenly divisible by $y$ and $y$ is not evenly divisible by $x$" one can obtain

$$\lambda xy[x \text{ is evenly divisible by } y \text{ and } y \text{ is not evenly divisible by } x].$$

In general, for any expression (form, formula, term, definite description, λ-abstract) $\Psi$ and variables $\ulcorner \alpha_1, \ldots \alpha_n \urcorner$, one can obtain

$$\lambda \alpha_1, \ldots \alpha_n[\Psi \alpha_1, \ldots \alpha_n].$$

All such expressions are compound function expressions.

However, instead of forming compound *n*-adic λ-abstracts that represent *n*-ary functions (or functions of $n$ variables), Church iterates monadic "λ" and reduces *n*-ary functions to functions of one variable whose values are functions of $n-1$ variables. (This is the method known today as "Currying", for Haskell Curry, although the method is due to Moses Schönfinkel; see Church 1941 [BE: 202–3].) Rather than writing "$x+y$" for the function from pairs of natural numbers to their sum we write "$((+x)y)$". For any number $x$, we want "$+x$" to designate the function of one variable that when applied to the value of this variable gives another function $(+x)$ that for any number $y$ gives $((+x)y)$, i.e., the sum of $x$ and $y$, and thus adds $x$ to any number $y$. In this way, we input each variable into a separate function. Rather than using *n*-adic "λ" and writing

$$\lambda xy[x+y]$$

we use monadic "λ" and write

$$\lambda x(\lambda y[+x]y).$$

In this notation "$\lambda x(\lambda y[+x]y)$" *designates* the function of one variable that when applied to the value of "$x$"—say 1—as argument gives another function designated by "$\lambda y[+1]y$" that for any number $y$ adds 1 to its argument. Since the latter function is itself a function of one variable, we have reduced the function $\lambda xy[x+y]$ of $n=2$ variables to a function of one variable whose value is a functions of $n-1$ variables.

Having reduced *n*-adic λ-abstraction to iterated monadic λ-abstraction, for any expression (form, formula, term, definite description, λ-abstract) $\Psi$ and variable $\ulcorner\alpha\urcorner$, we can write

$$\lambda\alpha[\Psi\alpha]$$

instead of

$$\lambda\alpha_1,\ldots\alpha_n[\Psi\alpha_1,\ldots\alpha_n].$$

Church sometimes adopts Russell's notational convention (PM) that a dot placed before an expression brackets the expression. "$\lambda\alpha[\Psi\alpha]$" is then the same expression as

$$\lambda\alpha\,.\,\Psi\alpha.$$

In general, this should be read as *the function that assigns $\Psi$ to $\alpha$*. When attached to a term, such an expression forms a compound monadic predicative formula, as in

$$\lambda y\,.\,{+}1y\;2$$

$\lambda x\,.\,x$ is a Polish diplomat and $x$ is a great pianist *Paderewski*.

In general, when $\ulcorner\lambda\alpha\,.\,\Psi\alpha\urcorner$ is combined with an argument expression $\beta$, it should be read as *the value of the function that assigns $\Psi$ to $\alpha$ for $\beta$ as argument*.

The elegance of the notation as a means for representing functions is obvious. The identity function, which is defined as $I\alpha=\alpha$ for any argument $\alpha$, is represented by the abstract $\ulcorner\lambda\alpha\,.\,\alpha\urcorner$. The function that duplicates or replicates its own argument is represented by the abstract $\ulcorner\lambda\alpha\,.\,\alpha\alpha\urcorner$ (Church 1941 [BE: 202]). The constant function, which accepts α as well as any second argument $\beta$ but always maps α to the same value regardless of $\beta$, is represented by the abstract $\ulcorner\,{:}\,\lambda\alpha\,.\,\lambda b\alpha\urcorner$. (Note that Church sometimes adopts Russell's notational convention (PM) that a greater number of dots indicates an outside bracket, while a smaller number indicates an inside bracket.)

λ-abstracts are governed by three rules of inference:

- $\alpha$-conversion: alphabetic changes of bound variables
- λ-contraction: an occurrence of $\ulcorner \lambda\alpha \,.\, \Psi\alpha\beta \urcorner$ within a formula is replaced with one of $\Psi\beta$
- λ-expansion: an occurrence of $\Psi\beta$ within a formula is replaced with one of $\ulcorner \lambda\alpha \,.\, \Psi\alpha\beta \urcorner$.

Regarding the last two rules, it is required that Ψβ be the result of uniformly substituting free occurrences of $\beta$ for free occurrences of α throughout Ψα. These two rules are together known as "λ-conversion" (although "λ-contraction" is sometimes known as "$\beta$-reduction" or "λ-concretion" and "$\alpha$-conversion" is sometimes known as "$\alpha$-reduction".) For example, from

Paderewski is a Polish diplomat and Paderewski is a great pianist

we can infer by λ-conversion

$\lambda x \,.\, x$ is a Polish diplomat and $x$ is a great pianist *Paderewski*,

and vice versa. Similarly, from "$\lambda x \,.\, x\ a$" we can infer "$a$" and vice versa, while from "$\lambda x \,.\, xx\ a$" we can infer "$aa$" and *vice versa*. Likewise, from "$1 + 2$", i.e., "$((+1)2)$" we can infer

$$(\lambda y[+1]y)2$$

$$\lambda x(\lambda y[+x]y)1, 2$$

and *vice versa*. Using our preferred notational convention:

$$\lambda y \,.\, +1y2$$

$$\lambda x : \lambda y \,.\, +xy12.$$

The process of λ-contraction terminates iff it results in an expression in normal form, where an expression is in *normal form* iff neither it nor any part of it is of the form $\ulcorner \lambda\alpha \,.\, \Psi\alpha\beta \urcorner$. For example, "$aa$" is the normal form of "$\lambda x \,.\, xxa$" while "$1 + 2$" is the normal form of "$\lambda x : \lambda y \,.\, +xy12$". Some expressions such as "$:\lambda x \,.\, xx : \lambda x \,.\, xx$" have no normal form; in this case applying λ-contraction to the expression yields the very same expression of the form $\ulcorner \lambda\alpha \,.\, \Psi\alpha\beta \urcorner$, so the process of λ-contraction does not terminate.

The above rule of alpha-conversion already allows that two distinct expressions—say, "$\lambda x \,.\, xa$" and "$\lambda y \,.\, ya$"—can represent the same function. Much more interestingly, from each of "$\lambda x \,.\, xa$" and "$:\lambda y \,.\, \lambda x \,.\, xya$" we can, by λ-conversion, infer $a$, for any argument $a$; however, by the above rules we *cannot* infer "$\lambda x \,.\, x$ " from "$:\lambda y \,.\, \lambda x \,.\, xy$". (It follows that if we restrict ourselves to the above rules, then these two expressions are not synonymous; cf. supplement E.) So we have *two distinct λ-expressions that represent distinct functions with the same graph*, which illustrates Church's innovation of representing functions as rules of correspondence (or functions in intension) rather than as graphs (or functions in extension).

However, the following are special cases of λ-contraction and expansion respectively, if Ψ is a monadic predicate or function symbol not containing any free occurrence of α:

- eta-contraction: an occurrence of $\ulcorner \lambda\alpha \,.\, \Psi\alpha \urcorner$ within a formula can be replaced by one of $\Psi\alpha$
- eta-expansion: an occurrence of $\Psi\alpha$ within a formula can be replaced by one of $\ulcorner \lambda\alpha \,.\, \Psi\alpha \urcorner$.

If we allow eta-contraction as a special case of λ-contraction, then we *can* infer "$\lambda x \,.\, x$" from "$:\lambda y \,.\, \lambda x \,.\, xy$". More generally, the theory becomes extensional because two closed expressions in normal form are distinct iff they designate functions that give different values for some argument and so have distinct graphs. Computable functions with the same graph are no longer distinct. For more information, see Hindley and Seldin (2008: ch. 7).

Another difference between functions and their graphs is that in an un-typed λ-calculus compound function expressions can be applied to themselves. For example, we can apply the identity function to itself, as in "

$:\lambda x \,.\, x : \lambda x \,.\, x$", and infer "$\lambda x \,.\, x$". We can also apply the constant function to a first argument $a$ and to itself as the second argument, as in "$:\lambda x \,.\, \lambda y x a : \lambda x \,.\, \lambda y x$", and infer "$a$". However, if we were to identify such functions with their graphs, we would have to allow graphs that can contain themselves—something disallowed in conventional set theory by the axiom of foundation.

Given that compound function expressions can be applied to themselves, we can obtain a formula that purports to designate the Russellian function that does not apply to itself: "$\lambda \mathrm{F} : \neg \mathrm{F} \,.\, \mathrm{F}$", i.e., "$\lambda \mathrm{F}(\neg \mathrm{F}[\mathrm{F}])$", which we shall call "$R$". The formula "$RR$" expresses that $R$ applies to itself. By the definition of "$R$" and λ-conversion:

$$RR \leftrightarrow \lambda \mathrm{F} : \neg \mathrm{F} \,.\, \mathrm{F}R$$

But then, by an application of λ-contraction to the right hand side, $RR \leftrightarrow \neg \,.\, RR$. However, Church originally developed the λ-calculus within an un-typed deductive system that was supposed to provide a foundation for mathematics that avoided this paradox (1932, 1933). Neither $RR$ nor $\neg \,.\, RR$ can be derived in the system of 1933, so the paradox cannot be obtained.

Numerals and arithmetical operations can be represented in the λ-calculus. Church begins with the numeral "1" (1941 [BE: 219]); but it is now common to begin with "0" and we do so here. We begin with an arbitrary function $f$. To represent "0" we form a λ-abstract that designates $f$ as well as the identity function with $f$ unapplied:

$$0 =_{df} \lambda f \,.\, \lambda x \,.\, x$$

Intuitively speaking, the connection between the designatum of this abstract and the familiar number 0 is that the identity function with $f$ unapplied will always return *x with nothing added to it,* for any argument $x$.

To represent 1 we designate the same function as before but with $f$ applied:

$$1 = \lambda f \,.\, \lambda x \,.\, fx$$

If $g$ and $h$ are functions, then there is another function that results from applying $g$ to the result of applying $h$. This is designated by "$\lambda x \,.\, g(hx)$" and is also called "the composition of $g$ and $h$". For example, where the function $h = \lambda x \,.\, x^2$ and the function $g = \lambda x \,.\, x + 1$, their composition $= \lambda x \,.\, x^2 + 1$ (i.e., first we square the value of $x$ and then we add 1 to the result). To represent 2 we designate the function that when applied to $f$ as argument gives the composition of $f$ and itself:

$$2 = \lambda f \,.\, \lambda x \,.\, f(fx)$$

We iterate this method to represent the rest of the natural numbers:

$$3 = \lambda f \,.\, \lambda x \,.\, f(f(fx))$$
$$m = \lambda f \,.\, \lambda x \,.\, f \ldots . (f(fx)).$$

As noted in Supplement A, these numerals as well as the notion of normal form are essential to Church's basic undecidability argument.

The forgoing explanation of "the composition of $g$ and $h$" took the function $x + 1$ for granted. This too can be represented in in the λ-calculus:

$$S(m) =_{df} \lambda y \,.\, \lambda f \,.\, \lambda x \,.\, f(yfx)m$$

For example, by this definition and λ-conversion

$$\begin{aligned} S(0) &= S(\lambda f \,.\, \lambda x \,.\, x) \\ &= (\lambda y \,.\, \lambda f \,.\, \lambda x \,.\, f(yfx))(\lambda f \,.\, \lambda x \,.\, x) \\ &= \lambda f \,.\, \lambda x \,.\, fx \\ &= 1. \end{aligned}$$

As Church remarks, this identification of the natural numbers with certain functions is allowable because the functions form a progression. Moreover, the arithmetical operations can also be represented in the λ-calculus (Church 1941 [BE: 220]).

For further discussion of Church's untyped λ-calculus see section 5. (See also Kleene [1981] as well as the entry on the lambda calculus and the section on λ-definability in the entry on Turing Machines.)

## D.2 Church's Simple Theory of Types

There are two basic observations behind the theory of types, one of which accords well with mathematical practice and the other with common sense. The first observation is that numbers, sets of numbers, sets of sets of numbers, functions, sets of functions, etc., are entities of different types, where types themselves are reflected in the language. This observation accurately reflects the practice of mathematicians who already know the types of the entities that they are thinking about when working in one or another subject area and who will draw type distinctions when working in a more general framework, like set theory, by using different kinds of symbols to designate numbers (e.g., "1", "2", "3"…, ), sets of numbers (e.g., "$\mathbf{N}$") and sets of sets of numbers (e.g., "$\mathcal{P}(\mathbf{N})$"). The second observation is that entities of a given type apply to entities of the highest type below their type, so do not apply to entities of their own type. This can be seen from the fact that while the expressions "Paderewski is over 5 feet tall", "Someone is over 5 feet tall", and "It is commonplace to be a person over 5 feet tall" are intelligible, "Everyone Paderewski", "Everyone someone", and "is over 5 feet tall is a commonplace quality to have" are all nonsense.

However, Church formulates his simple theory of types (STT) in a way that uses the λ-calculus to represent the typed entities as functions *qua* rules of correspondence rather than as sets or classes (cf. previous section). All expressions of STT are assigned types, which are the types of the entities that they designate. "$\iota$" symbolizes the type of individuals, "$o$" the type of truth values, and if α and β are type-symbols, then $\ulcorner(\alpha\beta)\urcorner$ is the type-symbol of the type of functions from entities of type α to those of type β. (Church writes this $\ulcorner(\beta\alpha)\urcorner$.) For example, "$(\iota o)$" symbolizes the type of functions from individuals to truth values, and "$(oo)$" symbolizes the type of functions from truth values to truth values.

According to STT, well-formed expressions are limited to:

i. Variables and constants of type α, for example "$x_\iota$", "$P_o$", and "$\neg_{(oo)}$".
ii. Complex expressions of type β formed from expressions of type α and those of type $(\alpha\beta)$, for example "$\neg_{(oo)}P_o$"
iii. λ-abstracts of type $(\alpha\beta)$ formed from variables of type α and expressions of type β

As a result of these restrictions, in STT there is no type of expressions δ formed only from expressions of type δ. It follows that $R$—"$\lambda F : \neg F \,.\, F$", i.e., "$\lambda F(\neg F[F])$"—is not well formed, and so that Russell's paradox cannot arise.

Church himself did not pursue the investigation of his STT in great depth, but two remarkable students of his did: Leon Henkin and Peter Andrews. Henkin developed what is now understood to be the model theory for higher order logic. He introduced the distinction between standard and non-standard models of STT and proved a completeness theorem for validity defined in terms of non-standard models; although, as a consequence of the incompleteness theorems, no such theorem is possible for validity defined in terms of standard models (see the entry on Church's type theory for some details). Andrews first turned his attention to extending STT to transfinite types, which allows for a more uniform development of number and cardinality than in STT with only finite

types; he then developed the proof theory of STT with an emphasis on automated reasoning. STT has inspired a lot of work on automation as well as on the application of STT to natural language semantics, mathematics and computer science, which is described in the entry on Church's type theory. (By the way, Church [1960, 1962a] also made important contributions to the theory of finite automata.)

Regarding the denotational semantics for STT, this is considerably more straightforward than that of un-typed λ-calculi. See Andrews (1986), which contains a lot of information about STT, much of which is also contained in the entry on Church's type theory.

The expressive power of STT raises an important issue that pertains to Church's objection to logicism as an economical foundation for elementary arithmetic (an objection noted in section 5):

> However, the higher-order predicate (or functional) variables, together with comprehension principles which are required for them, mean in the presence of an axiom of infinity that even the non-denumerably infinite has been admitted. (1962 [BE: 611])

To understand this objection, it will be helpful to recall that the expressive power of second-order logic is obtained by laying down comprehension axioms, which are axioms stating that a formula $\Phi$ defines a second-order entity in the domain of the higher-order variables, such as a propositional function, or the characteristic function of a class. It will also help to recall that these axioms have the following form:

$$\exists P \forall x_1 \ldots x_n \, . \, P x_1 \ldots x_n \leftrightarrow \Phi x_1 \ldots x_n$$

Now suppose that we want to define *complex* properties and functions of individuals (so we are staying at the second order). To do this we use the λ-abstraction operator discussed above. Then, by the axiom above, and recalling that in Church's system *n*-ary functions reduce to particular cases of functions of one variable, we can abstract the corresponding complex predicate, as follows:

$$\lambda x \, . \, \Phi x.$$

In the context of STT we need to proceed beyond second order logic, so we need a more general comprehension axiom scheme stating that an expression $\Psi_\beta$ of any type $\beta$ defines a complex function $u$ of type $\ulcorner(\alpha\beta)\urcorner$, which does not occur free in $\Psi_\beta$:

$$\exists u_{(\alpha\beta)} \forall x_a \, . \, u_{(\alpha\beta)} x_\alpha \approx \Psi_\beta$$

(Since $\Psi_\beta$ is of any type β, we use boldface "$\approx$", which reduces to "$\leftrightarrow$" in case $\Psi$ is a formula, and to identity in case $\Psi$ is a singular term. Further, since we now use lowercase Greek letters as variables ranging over *types*, we use "$x$", "$u$", "$y$" etc. as meta-language variables.) We designate the function whose existence is asserted directly above as

$$\lambda x_\alpha \, . \, \Psi_\beta x_\alpha$$

So we have the comprehension scheme

$$\forall x_\alpha : \lambda x_\alpha \, . \, \Psi_\beta x_\alpha \approx \Psi_\beta$$

More perspicuously, we can designate the aforementioned function

$$\lambda x_\alpha \, . \, \Psi_\beta$$

So the comprehension scheme becomes

$$\forall x_\alpha : \lambda x_\alpha \, . \, \Psi_\beta \approx \Psi_\beta$$

Corresponding to this axiom scheme, the λ-conversion rule of λ-expansion licenses the replacement, within a formula, of any occurrence of $\Psi_\beta$ by the λ-abstract $\ulcorner \lambda x_\alpha \,.\, \Psi_\beta y_\alpha \urcorner$. The rule of λ-contraction licenses the reverse replacement. Notice that that $x_\alpha$ and $y_\alpha$ are required to be of the same syntactic type.

Church's worry about logicism can be appreciated by asking the following question. *What are the possible values of the higher-order variables in the aforementioned comprehension axioms, under their classical extensional interpretation*? To which the usual answer is that any given second-order variable "*F*" ranges over *all sub-classes* of the domain of individuals that "*F*" is true of. Further, any axiom of infinity implies that there are infinitely many individuals. In which case, any given second-order variable "*F*" ranges over *all sub-classes* of an infinite class, which is equivalent to the powerset of natural numbers $\mathcal{P}(\mathbf{N})$. Thus, as Church remarked, "the non-denumerably infinite has been admitted" (1962 [BE: 611]). This discredits logicism as an economical foundation for elementary arithmetic.

More important for Church's purposes than the status of logicism is the relationship between the Simple and the Ramified theories of types as described in Church (1976).

## D.3 The Ramified Theory of Types

The motivation for ramification is to resolve all paradoxes: not only Russell's but also the semantical paradoxes such as Grelling's Paradox (GP) and the Russell-Myhill Paradox (RM), which are discussed in section 4. (Other semantical paradoxes addressed by ramification are those of Epimenides and Richard.) Although these semantical paradoxes do not arise in STT *per se*, they do arise in STT with the addition of a semantical predicate. Further, they can be solved—as Russell (1905; PM) discovered—by further stratifying each type. Intuitively, the idea is as follows:

Individuals have certain attributes such as *having brown hair*. Suppose that some attributes are such that *all* the children of a person who has these attributes also have them. These attributes are *hereditary*, and so we discern the attribute of attributes of *being hereditary*. Further suppose that some individuals have *all of* the aforementioned person's hereditary attributes. Here, "all" quantifies over attributes that have the attribute of *being hereditary*. Finally, suppose that we define the aforementioned person's *descendants* as the individuals (including that very person) who possess all of her hereditary attributes. As Poincaré (1906) and Russell (1905; PM) pointed out, such "impredicative" definitions display a kind of circularity. In the present case, we have helped ourselves to the notion of *all of her hereditary attributes* to define the notion of *her descendants* where *being her descendants* is itself one of her hereditary attributes. From the perspective of STT, it is perfectly legitimate to define expressions of lower type by using expressions of higher type. However, as we will see, from the point of view of ramified type theory (RTT) such definitions are illegitimate. (See also the entries on Poincaré and on Russell's paradox.)

Although it is helpful to have a prior understanding of STT in order to understand RTT, strictly speaking the type-theoretic notions involved are different. For this reason we adopt Church's policy of renaming the types "*r*-types" and using slightly different notation. Individual variables and names are assigned the type "$i$". Expressions are also assigned *levels*, represented by Hindu-Arabic numerals. Individual variables, names of individuals and names of propositions are of level 0. As for function expressions, these are assigned *r*-types as follows. If β is an *r*-type symbol, then $\ulcorner(\beta)/n\urcorner$ is the *r*-type symbol of all function expressions of level $n$ (where $n > 1$, because only individuals and propositions are of level 0). For example, a function expression that takes individual variables and names, as well as functions, as its arguments has the *r*-type $(i(i))/n$. Further, $\ulcorner(\alpha)/k\urcorner$ (where $k < n$) is of the level *directly lower than* $\ulcorner(\beta)/n\urcorner$ if $\ulcorner(\alpha) = (\beta)\urcorner$.

More generally (where $m \geq 0$), if $\ulcorner\beta 1, \ldots \beta m\urcorner$ are *r*-type symbols, then $\ulcorner(\beta 1, \ldots \beta m)/n\urcorner$ is the *r*-type symbol of all (m-place) function expressions of *level* n (where $n > 1$). Further, $\ulcorner(\alpha 1, \ldots \alpha m)/k\urcorner$ (where $k < n$) is of the level *directly lower than* $\ulcorner(\beta 1, \ldots \beta m)/n\urcorner$ if $\ulcorner\alpha 1 = \beta 1\urcorner$,... $\ulcorner\alpha m = \beta m\urcorner$. For $\ulcorner(\beta 1, \ldots \beta 3)/n\urcorner$, Church uses the abbreviation "$3/n$" and, more generally, $\ulcorner(\beta 1, \ldots \beta m)/n\urcorner$, is abbreviated "$m/n$".

To return to our example, "$i$" is attached to the individual variables "$x$", "$y$", that range over our descendants; these variables are also of level 0. By the above, the function expression "$x$ has brown eyes" is *r*-typed "$x$ has brown eyes"$^{(i)/1}$. Similarly, "$x$ is a descendant of $y$" is *r*-typed "$x$ is a descendant of $y$"$^{(ii)/1}$. Levels are *cumulative,* in the sense that the range of variables of each *r*-type includes the range of every variable of directly lower *r*-type.

Next, *orders* are defined as follows. The order of individual variables and names is 0. In the case of function expressions, the order of a variable of *r*-type $\ulcorner(\beta 1, \ldots \beta m)/n\urcorner$ is *the sum of n and the highest of the orders of the variables* $\ulcorner\beta\beta m\urcorner$ (reflecting that the levels are cumulative).

Now we can put all this together and apply it to our original example. In an *un*-typed language we can define *F is hereditary* as follows:

$$\text{Hereditary}(F, R) \equiv_{df} Fx \supset_x Rxy \supset_y Fy$$

(Note the notational convention that a connective can take a variable as subscript: that variable becomes a universal quantifier whose scope is precisely that of the connective.) In the present framework, since "$x$" and "$y$" are of level 0, while "$F$" and "$R$" are of level 1, "$F$" and "$R$" are also of order 1 (i.e., the sum of 0 and 1). Further, since Church's notational variant of "$\forall$" takes a name of a proposition as its argument, it is also of level 1 and order 1. Furthermore, there are no other argument expression in the definiens that are of a higher order than 1. Therefore, the level of $\text{Hereditary}(F, R)$ is 2 and its order is 3, which is the sum of 2 (its level) and 1 (the greatest of the orders of its arguments).

Finally, there is the restriction that for a function expression to be well formed, the highest order of any expression in the *definiens* must be of an order directly lower than the level of the *definiendum*. While this does not prohibit our definition of "$\text{Hereditary}^{2/2}(F^{1/1}, R^{2/1})$", it does prohibit using it in the aforementioned "impredicative" definition of "$x$ is a descendant$^{1/1}$", since this function expression is of level 1. As will become clear in section 4.3 and section 4.4, the solutions to GA and RM emerge from the apparatus of ramified types.

Crucially, if we drop the level indicators entirely, then the orders of a given function expression become simply the highest of the orders of its arguments. In that case, the orders become simple types, i.e., the "orders" of contemporary "first order" and "higher order" logic. The reason is that in first order logic, the highest of the orders of the variables (i.e., of "$x$" "$F$", "$R$" etc.) is 1. Further, in second order logic, the highest of the orders of the variables (ranging over attributes of attributes such as *being hereditary*) is 2. Note that the resulting simple type theory in Church's (1974b) is not formulated in a λ-calculus. (For more on Church's formulation of RTT, see the entries on the notation for *Principia Mathematica* 5.2 and on *Principia Mathematica* 4.1.2.)



Library of Congress Catalog Data: ISSN 1095-5054