# A Formal Introduction to the System of the *Principia Mathematica* of Whitehead and Russell

M. Randall Holmes

December 13, 2025

## Contents



## 1 Introduction

The subject of this essay is the introductory and explanatory material in the *Principia Mathematica* of Whitehead and Russell ([**?**]). Hereinafter, I will refer to the work as PM and to the author as Russell. This work owes substantial content to work of Kamareddine, Nederpelt and Laan in [**?**], though my approach is not the same.

Reading PM is maddening. The introductory material before the formal development starts are in terms of propositional functions rather than classes and relations as in the text. Only the most sketchy indications of the notation

for propositional functions are given. On p. 19, Russell says that he manages to keep use of propositional functions as constants or variables almost entirely out of the work. He is very good at this. But the place of propositional functions in the intellectual foundations of the work is crucial. In PM, Russell wants to present a theory of classes and relations which is in fact an impredicative simple type theory (as Ramsey pointed out later). But he is not confident of the validity of the concept of class, so he chooses to explain it in terms of the concept of propositional function. The fact that his description of the notion of propositional function and even of the simplest features of the notation for propositional functions is so very sketchy really impairs this project. Further, he is not comfortable with impredicative class or relation comprehension, and in fact requires that definitions of propositional functions be predicative. He uses the axiom of reducibility to justify impredicative reasoning, on which more below.

However, we believe that it is possible to extract from the hints a quite precise notion of what propositional functions are and indeed a fairly complete description of what notation for propositional functions must be like. We are helped in this by the fact that Russell's writing is always very precise, though it is often necessary to hunt around for clarifying remarks and examples.

A further point which has been addressed by others many times is that PM introduces an elaborate theory of types while having no notation for types at all! We make good on this by adopting notations developed long ago by others. Yet further, there are two different type schemes in the work. We adopt the first one (the scheme given in the introduction to the first edition) because the second one (in section 12 of the work and in the introduction to the second edition) is essentially useless without the axiom of reducibility to collapse its complexities. Russell makes an effort in the introduction to the second edition, where he has abandoned reducibility, to recover the capabilities of the former type system in the latter, but it appears to us that the type system of section 12, as elaborated further in the introduction to the second edition, is not really workable.

## 2 Propositional Functions

Russell's remarks on what propositional functions are and how they are to be notated are seemingly vague and scattered. As we will see, there is enough material in the book to put together a quite precise formal description of notation for propositional functions, in spite of this appearance. I am indebted for this to the prior work of Kamareddine, Nederpelt, and Laan ([**?**]). On p. 40, Russell tells us that the propositional function determined by an expression $\phi x$ containing a real variable $x$ should be written $\phi\hat{x}$. In the footnote on p. 40 he tells us that he will speak of values for $\phi x$ (open sentences) and values of $\phi\hat{x}$ (actual propositional functions), in each case meaning the same thing, specific sentences $\phi a$ in which the undetermined variable has been specified.

We follow Kamaraddine et. al. in allowing ourselves the abbreviation pf for

"propositional function".

His treatment of propositional functions of more than one variable is laid out on p. 200: where a sentence $\phi xy$ is converted to a propositional function $\phi\hat{x}\hat{y}$, the convention is that the first argument of the function will be the one with the hatted variable earlier in the alphabet. So $\hat{a} < \hat{b}$ is the relation "less than", and so is $\hat{b} > \hat{a}$.

It is also important to note that in the absence of head binders, all variable arguments of a function must actually occur free in the expression defining the function.

The important observation here is that in spite of the lack of explicit examples, this is not vague at all. This description, combined with an inventory of the primitive constructions of propositions in PM, allows us to give a complete description of how propositional function notation must work. There are considerable difficulties, though. They do not have to do with the definition, which is quite clear in its essentials, but with the problem that this notation binds variables without providing a head binder. Russell nowhere remarks on this directly, but he gives enough information to give a complete solution nonetheless.

## 2.1 The Formal Problem with Propositional Function Notation

On p. xxviii he gives an example which might suggest the need for a certain kind of constant (or perhaps of another sort of variable). He considers a formula $\phi!x$ (we will explain the exclamation point on the pf variable later) and introduces the letter $a$ as a constant, then (on the next page) invites us to consider $\phi!a$ as a function of $\phi$, the function being denotable by $\hat{\phi}!a$. Kamareddine (and I in [**?**]) drew the conclusion that we needed individual constants in addition to individual variables to handle this; here we will not in fact need to do this.

Constants or free variables (however one construes them) which appear in pf notations cannot be bound, due to the formal limitations of the notation. We will concern ourselves later to show that nonetheless the notation is adequate in principle to found all of Russell's work (it is not clear to us that Russell was aware of the consequences of his definition at all!).

We cannot view a sentence involving a pf $Ra\hat{x}$ (such as $Ra\hat{x} = Ra\hat{x}$) as a function of $a$ (we cannot circumflex the parameter $a$ which is left free) because the resulting $R\hat{a}\hat{x} = R\hat{a}\hat{x}$ is quite a different creature, a function of two variables $a$ and $x$ returning a proposition as a value, not a function of one variable $a$ returning a proposition as a value (and it says nothing about the object $Ra\hat{x}$ at all). And further, we cannot quantify over a variable $a$ in such a context, such as $(a)(Ra\hat{x} = Ra\hat{x})$, because any expression $(a)\phi a$ is to be understood as a function of $\phi\hat{a}$, and the function $R\hat{a}\hat{x} = R\hat{a}\hat{x}$ would have unintended structure. Any variables used in this way are really to be thought of as constants (as Russell says when he introduces one in the example above); they can be regarded as implicitly universally quantified over in the largest context (so ultimately as variables) but they cannot explicitly be bound. There is an example of such

a notation $Ra\hat{x}$ (really in this case $R\hat{x}a$), namely $\hat{x} = a$ on page 19, which he gives as an example of a pf constant.

So the convention of variable binding in pf notation (and in quantified notations involving pf notation) is fixed by Russell's language, even though he has not given us many examples of such notation. Any variable appearing free (uncircumflexed) in a pf cannot be bound by any variable binding construction in which it appears. Our warrant to introduce such free variables or constants is given because he has given examples of this (and because it is quite clear that such functions exist: $\hat{x} < 3$ is needed just as much as $\hat{x} < \hat{y}$, or , if one does not like the mathematical example because 3 is not an individual, ($\hat{x}$ loves Socrates) is needed just as much as ($\hat{x}$ loves $\hat{y}$)). The fact that they cannot be bound at all follows ineluctibly from his presentation of the pf notation without head binders.

A formal difficulty must be noted here. Russell states as a primitive proposition that wherever there is a proposition $\phi a$ there is a pf $\phi\hat{x}$. For the moment we must restrict this proposition to situations $\phi a$ where the term $a$ does not occur as a proper constituent of a pf notation – which does not restrict what Russell does anywhere in his work, as he avoids using pf notations! Proposition 9.15 must be restricted as indicated; there may be other similar situations. On reasonable assumptions, this restriction on abstractions can actually be lifted in effect (if pfs are taken to be extensional and therefore describable).

## 2.2 Notation for Propositions, and thus for Propositional Functions

To determine what pfs there are, it remains only to determine what propositions there are. We define propositional notations and pf notations recursively, simultaneously with notions of free variables in a propositional notation and quantified variables in a propositional notation.

First, one needs to determine what atomic propositions there are. We claim that the text supports the view of Kamareddine et. al. that the only atomic propositions are of the form $\phi(t_1, \ldots, t_n)$ where $\phi$ is a propositional function variable or $R_n(t_1, \ldots, t_n)$ where $R_n$ is an elementary function (a constant with indicated arity $n$). The latter, like any constants, will not be bound, though they may ultimately be viewed as universally quantified in the largest context. It is quite clear that Russell thinks that there are constant elementary predicates of individuals in his language. We believe that he absolutely needs constant elementary predicates of all predicative types in order for the axiom of reducibility to make sense, so we add this formal feature to his language (in fact, we allow elementary function constants in all locally predicative types; we see no reason to exclude nonpredicative types from the argument lists, and of course the type of a pf built directly from an elementary function is locally predicative because there is no quantification in its definition to give an unexpected boost to its order).

There are also propositional variables, which are also a species of atomic propositional notation. Propositional variables are always free (there are no

quantifiers over propositions).

There is no need for pf notations to appear in applied position at all. The result of replacing $\phi$ in $\phi!b$ with $\hat{x} = a$ (and so the value of $\hat{\phi}!b$ at $\hat{x} = a$) is not something like $[\hat{x} = a]b$, but simply $b = a$. The result of substitutiing a propositional function $\phi\hat{z}$ for the variable $x$ in the context $x(t_1)$ is the result of replacing $z$ with $t_1$ in $\phi z$. There is language explicitly supporting such a view in PM, and the consequences have been worked out formally in [**?**] and in a different way in my [**?**].

The terms $t_i$ will be either constants or variables of types determined by the type of $\phi$ or $R_n$ (for Russell the type of $R_n$ is precisely determined by $n$; we should really use notation $R^\tau$ because the type information we allow about such a predicate constant is more complex than just how many individuals it takes as arguments). Constant individuals we have already considered; constants of pf types will simply be pf notations. The form of variables in argument lists is an interesting feature of the notation of PM which we will review below. In [**?**] and [**?**], variables of all types were given as single letters, possibly with type information; we will below discuss exactly reproducing the notational scheme of PM.

The free variables in a notation $X(t_1, \ldots, t_n)$ will be $X$ if it is a variable rather than an elementary constant and whichever $t_i$'s are variables. We already know that a pf constant will not contain free variables eligible to be bound in larger contexts. There are no quantified variables in an atomic propositional notation (the effects on order of variables quantified in constituent pf constants will be recorded in the type of those pf constants, so we do not need to consider these).

The free variable in a propositional variable $p$ is $p$. There are no quantified variables in a propositional variable.

What composite propositions there are is straightforward to determine. We will follow the first edition (the approach of section 10) in saying that where we have propositional notations $P$ and $Q$ [without regard to their quantifier structure], we get propositional notations $\neg P$ and $P \vee Q$. The free variables in $\neg P$ are the free variables in $P$. The free variables in $P \vee Q$ are the variables free in $P$ or $Q$. The quantified variables in $\neg P$ are the quantified variables in $P$. The quantified variables in $P \vee Q$ are the variables quantified in $P$ or $Q$.

We follow the scheme of section 10 in the formal presentation of PM for defining quantified propositions. So we need only the primitive construction of universal quantification: if $P$ is a propositional notation in which $x$ is free and does not occur free in a pf notation, $(x)P$ is a propositional notation. The free variables in $(x)P$ are the free variables in $P$ other than $x$. The quantified variables in $(x)P$ are $x$ and the quantified variables in $P$. We can of course define $(\exists x)P$ as $\neg(x)(\neg P)$. The variable $x$ here is not a propositional variable.

From any propositional notation $P$ which contains a free variable we can construct pf notations by circumflexing some occurrences of free variables which do not themselves occur free in constituent pf notations of $P$ (not necessarily all of them). in this we differ from [**?**] or our earlier treatment in [**?**] where all variables were assumed circumflexed (so constants were required in the formalism

in place of the free variables we are allowing in pf notations) and in fact it was possible to omit using circumflexes entirely.

A propositional variable cannot be circumflexed.

We could refer to the variables which are blocked from being bound by appearing free in pf constants as "global variables". A propositional variable can be global.

## 2.3 Types and the Format of Variables and Argument Lists

We now review the type system. PM has no notation for types. We use a notation which has become standard (due to Ramsey? Church?). The type of individuals is denoted by 0. The type of individuals has order 0. Types of pfs are denoted by $(\tau_1, \ldots, \tau_n)^m$, where $m$ is the order of the type and is larger than all orders of types $\tau_i$. We follow the convention that the order $m$ can be omitted if it is as small as possible (one greater than the maximum order of a $\tau_i$).

In an atomic notation $X(t_1, \ldots, t_n)$ the type of $X$ must be of the form $(\tau_1, \ldots, \tau_n)^m$ where the type of each $t_i$ is $\tau_i$.

The type of a pf notation $P$ with circumflesed free variables $\hat{x}_1, \ldots, \hat{x}_n$ presented in alphabetical order will be $(\tau_1, \ldots, \tau_n)^m$ where $\tau_i$ is the type of $x_i$ for each $i$ and $m$ is the smallest natural number greater than the orders of all the types $\tau_i$ and the orders of all the quantified variables in $P$.

We may use the notation $\phi!(t_1, \ldots, t_n)$ when the order of the type of $\phi$ is one greater than the maximum of the orders of the types of the $t_i$'s. This is more general than the situation where Russell uses this notation. We say that the type of $\phi$ is locally predicative in this case, and we say that a type is predicative iff it is 0 or it is locally predicative and all its constituent types are predicative.

We now consider the forms of variables in PM. One may note that variables in applied position are given the form $\phi$ (or $\phi!$ to indicate predicative type, which is almost always the case) while variables in argument lists appear with argument lists of their own (with circumflexed arguments), as in $F!(\phi\hat{z}, x)$. In [**?**] and [**?**], Kamareddine et. al. and I wrote variables as atomic notations in both contexts. However, this has a significant effect on the allowed degree of polymorphism of typing of pfs, so I am resisting this temptation here.

Propositional variables are atomic and typographically distinct from other variables.

I should note further than in the cited works, no circumflexes were used. In fact, they were in a strictly formal sense not needed, because in those works *all* free variables in a propositional notation would be circumflexed to form a pf. Any occurrence of a propositional notation in a context appropriate to a proposition could be read as a proposition; any occurrence of a propositional notation in a context appropriate to a pf could then be recognized as such and so the fact that the variables in it were bound wouldl be formally determined by the context. It is crucially important for such an approach that the class of letters used as individual constants be distinguished from those used as variables; this

is not the case in PM, nor do we need to do this here, because we do distinguish between circumflexed and noncircumflexed variables. The distinction between circumflexed and noncircumflexed variables is important if one uses complex forms of variables as well.

The rule for forms of variables in argument lists in PM is that an atomic variable $x$ will be used in a pf argument list for an object of the lowest type under consideration (the type of relative individuals). The types of all atomic variables in the same argument list will thus be the same (but undetermined, except possibly by features of the larger context). Every other variable appearing in a given argument list will appear with an argument list (of circumflexed variables) indicating its type relative to the type of relative individuals. As in $F!(\hat{\phi}!\hat{z})$, an argument list might not actually contain any relative individual entries, but nested argument lists will eventually do so, as in this case, where $F$ has $\phi$ as its only formal argument, which in turn has the relative individual $z$ as its sole formal argument. Its order is fixed, in all examples given in PM, by the provision of an exclamation point: all arguments are assumed to be predicative. If its order is to be higher, the device of replacing the exclamation point with a subscript indicating the increment to its order could be used: $\phi_2\hat{x}$ could be used to denote an argument which takes a single argument of relative individual type but has order one higher than expected (due to some unrevealed quantification in its definition).

A circumflexed variable $\widehat{\phi\hat{z}}$ is quite a different creature from the pf of two arguments $\hat{\phi}\hat{z}$. There is I believe a place in PM where a formal mistake is made by writing one of these where the other must be intended.

These rules for variables allow some polymorphism, but much less than is permitted in my quite complex implementation of the PM type system in [**?**]. It should be noted that the precise effect of providing the formal argument lists can be given simply by requiring that argument variables have associated types, which are not their actual types but the result of replacing the type of relative individuals with 0 in their actual types. This does mean that the type of relative individuals must be a constituent of every constituent of the type of a pf for which that type of relative individuals is appropriate. Note that typing argument variables in this way relative to a type of relative individuals does allow specification of nonpredicative types simply by adding order superscripts. If one were to do formal work in the system of the second edition, where reducibility is abandoned, such features would be needed.

Notice that because of these devices, although no type superscripts are used, one does in fact have some type information about variables in argument positions. The failure to supply complete type information is deliberate: the system has polymorphism (or as Russell says, "systematic ambiguity").

## 2.4 Substitution Formalized

It is now necessary to give the formal definition of substitution. As Russell himself says, the notation $\phi b$ has no constituent $\phi$: the result of replacing $\phi\hat{z}$ here

with $\hat{x} = a$ is immediately $b = a$. This is important for a correct understanding of reasoning involving quantifiers over pfs.

Let a substitution be specified by a finite function $\sigma$ from variables to notations for individuals or pfs such that $\phi(x)$ is always of the same type as $x$. There may be propositional variables in the domain of $\sigma$, and $\sigma(p)$ will be a propositional notation if $p$ is a propositional variable.

$(x \to t)$ represents the substitution with single domain element $x$ and single domain element $t$.

$\sigma'(t)$ for any notation $t$ of any sort is $\sigma(t)$ if $t$ is a variable in the domain of $\sigma$ and $t$ otherwise. $\sigma'(t)$ is undefined if any domain element of $\sigma$ appears free in a pf notation in $t$ (if a domain element is a global variable in $t$).

We define $\sigma^*$, the substitution function on terms, for each propositional notation.

$\sigma^*(X(t_1, \ldots, t_n)) = \sigma'(X)(\sigma'(t_1), \ldots, \sigma'(t_n))$, unless $\sigma'(X)$ is a pf constant (which will only happen if $X$ is a variable; an elementary predicate symbol is fixed by $\sigma'$), in which case $\sigma^*(X(t_1, \ldots, t_n)) = \chi^*(\sigma(X)')$, where $\sigma(X)'$ is the propositional notation from which the pf notation $\sigma(X)$ is derived and $\chi$ maps the free variables $y_i$ in $\sigma(X)'$ which are circumflexed in $\sigma(X)$ (taken in alphabetical order) to the terms $\sigma'(t_i)$. This computation is guaranteed to terminate by induction on the complexity of types.

$\sigma^*(p) = \sigma(p)$ for a propositional variable $p$.

$\sigma^*(\neg P) = \neg\sigma^*(P); \sigma^*(P \vee Q) = \sigma^*(P) \vee \sigma^*(Q)$

$\sigma^*((x)P) = (z)\sigma^*((x \to z)^*(P))$, where $z$ is the first available variable not appearing free, quantified, or globally in $P$ or in any range value of $\sigma$. We admit that this clever device for avoiding bound variable capture is an anachronism, but Russell was certainly aware of the issue. (Notice that substitution for a propositional variable will cause bound variables to be moved so as not to capture any variable free in the proposition).

## 2.5 An Aside on Comprehension Principles

The crucial application of the formal definition of substitution is to correct understanding of the use of quantifiers over pfs. There is for example no need for a scheme of comprehension $(\exists A.(\forall x.x \in A \leftrightarrow \phi))$, ($\phi$ predicative, $x$ the only variable free in $\phi$) because $x \in A$ abbreviates $A(x)$ [this is a simplification, admittedly, but in the second edition it is literally correct] and $(\forall x.A(x) \leftrightarrow \psi)$ is instantiated by the pf obtained from $\psi$ by circumflexing $x$, because when this pf replaces the variable $A$, $A(x)$ simply becomes textually identical to $\psi$.

Notice that this only worked for parameter free comprehension. In fact we can prove instances of comprehension involving parameters, but only with some care. For example the argument for the existence of $A \cup B$ goes like this. $A$ and $B$ correspond to functions $\phi x$ and $\psi x$. So of course we can define $\phi\hat{x} \vee \psi\hat{x}$ in each case and obtain the function corresponding to $A \cup B$. Note that this requires us to be able to circumflex only some of the free variables in a propositional notation. The variables $\phi$ and $\psi$ cannot be quantified over in this context. This must be a use of rule 10.11, when properly formalized.

Let $A$ and $B$ be any classes. These can be implemented by pfs $\alpha$ and $\beta$ such that $x \in A$ means $\alpha x$, $x \in B$ means $\beta x$. We can then form the pf $\alpha\hat{x} \vee \beta\hat{x}$ which corresponds to a class which we can see has the right extension: this is what cannot be done in the system of [**?**] or my [**?**]. So we can conclude $(\exists C.(\forall x.x \in C \leftrightarrow x \in A \vee x \in B))$ in the particular case of the classes $A$ and $B$. So we can conclude $(\forall AB.(\exists C.x \in C \leftrightarrow x \in A \vee x \in B))$ by 10.11. Any parameterized instance of comprehension can be proved in this way.

The reasoning actually goes like this.

$\vdash (x)(\alpha(x) \vee \beta(x) \leftrightarrow \alpha(x) \vee \beta(x))$ [easily proved]

$\vdash \phi(u) \rightarrow (\exists w)(\phi w)$ [a logical principle]

instantiate $u$ with $\alpha\hat{x} \vee \beta\hat{x}$ and $\phi\hat{t}$ with $(x)(\alpha(x) \vee \beta(x) \rightarrow \hat{t}(x))$ to get

$\vdash (x)(\alpha(x) \vee \beta(x) \leftrightarrow \alpha(x) \vee \beta(x)) \rightarrow (\exists w)((x)(\alpha(x) \vee \beta(x) \leftrightarrow w(x)))$

and by modus ponens

$\vdash (\exists w)((x)(\alpha(x) \vee \beta(x) \leftrightarrow w(x))$

and by universal generalization

$\vdash (\alpha, \beta)(\exists w)((x)(\alpha(x) \vee \beta(x) \leftrightarrow w(x))$

This gives a general model for proving instances of comprehension with parameters. It also gives an example of reasoning which can only be carried out with the use of real variables.

## 2.6 The Axiom of Reducibility

This axiom asserts that $(\exists\phi)(x)(\phi!x \leftrightarrow \psi x)$ for any $\psi$ however impredicative. $(\exists\phi)(x)(y)(\phi!xy \leftrightarrow \psi xy)$ is the version for binary relations. More versions could be supplied if desired.

The effect of this axiom is to entirely remove the restrictions on comprehension imposed by the predicativity. This is well understood, having first been noted by Ramsey.

What interests us about it is something else, which is only evident on some careful examination of the text, because it is not something one would believe in a modern treatment even from a predicativist standpoint. Russell appears to mean by "predicative" here precisely "is defined without quantifiers over domains of order higher than those of its arguments" (this may be more liberal than what he says), which is actually a feature of the text of the propositional function $\phi$, as it were. He appears to actually be asserting that there will be a predicative term of his formal language equivalent to any nonpredicative term $\psi$ in one or two variables: this claim is likely to be objectively false unless one supposes that his language has unexpected resources (I provide mine with these in the form of a large supply of constant elementary functions of all predicative orders).

## 2.7 The Reduction of Classes to Propositional Functions

We recall the definition of $\phi x \equiv_x \psi x$ as $(x)(\phi x \leftrightarrow \psi x)$.

This is achieved by a contextual definition

$f\{\hat{z}(\psi z)\} \equiv_{\mathtt{def}} (\exists\phi.\phi!x \equiv_x \psi x \wedge f\{\phi!\hat{z}\})$.

I believe that this works under the proposed implementation of pfs. The expansion of this assertion inside a quantified context might make one queasy, but I think things will actually work as expected.

Relations are expanded in the same way.

# 3 About the Contextual Definitions

One of our aims in writing this essay is to support computer formalization of the system of PM, and Russell's contextual definitions of descriptions and classes are a serious problem in this respect, because the expansions of quite simple expressions are horrible and the issues of scope of context to be considered which arise are hideous. The simplest uniform principles of reasoning on these definitions involve elaborate recursive arguments (the justifications of which *are* provided in the text, it must be noted).

We propose somewhat different interpretations of the notations $(\iota x.\phi x)$ and $(\hat{x}.\phi x)$ which demonstrably have the same effect and are easier to support.

The notation $(\iota x.\phi x)$ is defined as meaning $\phi\hat{x} \wedge \phi y \equiv_y y = \hat{x}$. This is a pf true of precisely the desired object if it exists and otherwise of nothing. The crucial move is then that in any atomic sentence involving a descriptive term, we in fact interpret things setwise. If $d$ is a descriptive term, $f(d)$ is interpreted as $(\exists x).(d(x) \wedge f(d))$; $d\,R\,e$ is interpreted as $(\exists xy)(d(x) \wedge e(y) \wedge x\,R\,y)$, and similarly for atomic propositions with more arguments. Where a real term coexists with a descriptive term, interpret the real term $t$ as $(\iota x.x = t)$. This has precisely the desired effects. Atomic sentence involving descriptions that are not satisfied become false. The notation $E!d$ for "$d$ exists" is easily defined as $d = d$ or $d \in V$. For formal purposes, For uniform reasoning, it is probably useful to have free variables available which are marked as of descriptive type (PM does not have such a mechanism, but it would be handy for stating certain kinds of general results – such a variable $d$ would transform to $(\iota x.d'(x))$ for $d'$ an associated free pf variable).

For class notation, we suggest an approach similar to Russell's in spirit but not in fact involving a contextual definition. Define predicates $x \sim y$ over each type and predicates $y^*(\hat{x}_1, \ldots, \hat{x}_n)$ with the same typing as $y(\hat{x}_1, \ldots, \hat{x}_n)$ in all types. Define $x \sim y$ as $x = y$ for $x, y$ individuals. Define $u(\hat{y}_1, \ldots, \hat{y}_n) \sim v(\hat{y}_1, \ldots, \hat{y}_n)$ as holding iff $u^*(y_1, \ldots, y_n) \equiv_{y_1,\ldots y_n} v^*(y_1, \ldots, y_n)$. Define $y^*(x_1, \ldots, x_n)$ as $(\exists x'_1, \ldots x'_n)(x_1 \sim x'_1 \wedge \ldots x_n \sim x'_n \wedge y(x'_1, \ldots, x'_n))$. Define $x \in y$ as $y^*!(x)$. Define $x\,R\,y$ as $R^*!(x, y)$. One could either use $\sim$ to denote equality of classes and relations in extension, or take the more radical move of completely replacing $x(y_1, \ldots, y_n)$ with $x^*(y_1, \ldots, y_n)$ in all contexts, in which case $\sim$ becomes equality by definition, extensionality holds, and class notation becomes naturally definable. One also needs to verify that the interpretation using starred predicates supports the basic constructions of pfs (show for example that $\neg\phi^*\hat{x}$ is equivalent via $\sim$ to a starred predicate [in fact to itself starred]). This is actually formally very similar to applying Russell's contextual definition to all levels of the type hierarchy at once. It should be noted that once extensionality holds,

pfs with parameters become definable, and it becomes possible to quantify into pf notations indirectly.

An alternative approach would be to introduce a local contextual definition of classes and relations in extension in each type, similar to the handling of definite descriptions above. The notation $\hat{x}(\psi x)$ would denote the $\sim$ equivalence class of the predicative pfs coextensional with $\psi$ (and similarly for relations of whatever arity). All objects in an atomic proposition containing any class symbol would be interpreted in this way, and all pfs would be treated as if starred, and read setwise. This seems excessively complex to actually implement in a computerized formal system, however. What might be worth formalizing is the machinery for making an inductive definition on types as in the previous paragraph and the formal proofs of the assertions about pfs made there, justifying passing to a system with extensionality from one without (and such machinery could also be applied to Russell's original definition). Our approach might be construed as identical to Russell's but with a different scoping decision: expand everything as far as possible and treat every object as a class or relation in extension.

# 4 Primitive Propositions of a modified PM

This is a first draft of a list of fundamental propositions. I do state some revisions. In keeping with my section 10 approach, I do not admit typing of propositions at all. This will require careful attention to how propositional variables are to be understood. I impose the restriction on variable binding into pf notations. I note that he does not state an analogue of 9.15 for multiple arguments; this is needed. It is clear that in roughly the place where 9.14 and 9.15 appear, a full discussion of pf formation and substitution is needed. In my description of the types, I describe an even slightly liberalized version of the type system of the introduction.

I believe that the proper rule for a propositional variable is that it is always free, and it may be replaced by any expression not containing a variable bound in the context in which it appears. This is not a real restriction because bound variables can be renamed (and the system needs a formal rule for this). Further, a variable open sentence with given arguments may be replaced by a new variable open sentence with more arguments as long as the new arguments are not bound in the context. My substitution definition in the first part now handles propositional variables.

actually, extending the argument list of an open sentence is allowed already. In $\phi x$, replace $\phi\hat{u}$ with $\psi\hat{u}t$ to get $\psi xt$. The only special substitution rule needed is the one for propositional variables.

**assertion:** $\vdash p$ means that we assert $p$. Propositional functions (as open sentences) may be asserted as well.

**negation:** If $p$ is any proposition, $\neg p$ is a proposition.

**disjunction:** If $p, q$ are propositions, $p \vee q$ is a proposition.

**implication:** $p \to q$ is defined as $\neg p \vee q$.

**1.1** Anything implied by a true [elementary] proposition is true.

[We make no use of distinctions between types of propositions – MRH]

**1.11** When $\phi x$ can be asserted, where $x$ is a real variable, and $\phi x \to \psi x$ can be asserted, where $x$ is a real variable, then $\phi x$ can be asserted, where $x$ is a real variable.

This is also to be assumed for functions of more than one variable.

This is also apparently used for propositional functions of propositional variables in this opening section – MRH

**1.2** $\vdash (p \vee p) \to p$

**1.3:** $\vdash q \to (p \vee q)$

**1.4:** $\vdash (p \vee q) \to (q \vee p)$

**1.5:** $\vdash (p \vee (q \vee r)) \to (q \vee (p \vee r))$

**1.6:** $\vdash (q \to r) \to ((p \vee q) \to (p \vee r))$

**1.7, 1.71, 1.72** These propositions asserting closure of the elementary propositions under negation and disjunction are not used, as we do not use the notion of elementary proposition – MRH.

**3.01:** $p \wedge q$ is defined as $\neg(\neg p \vee \neg q)$

**4.01:** $p \leftrightarrow q$ is defined as $(p \to q) \wedge (q \to p)$

**universal quantifier:** If $\phi\hat{x}$ is a propositional function, $(x)\phi x$ is a proposition.

**existential quantifier:** $(\exists x)(\phi x)$ is defined as $\neg(x)(\neg\phi x)$

**formal implication:** $\phi x \to_x \psi x$ is defined as $(x)(\phi x \to \psi x)$

**formal equivalence:** $\phi x \equiv_x \psi x$ is defined as $(x)(\phi x \leftrightarrow \psi x)$

**9.1** $\vdash \phi x \to (\exists z)(\phi z)$

**9.11:** $\vdash \phi x \vee \phi y \to (\exists z)(\phi z)$

**9.12:** What is implied by a true premise is true (i.e, from $\vdash p$ and $\vdash p \to q$ we may proceed to $\vdash q$, and similarly for propositional functions of one or more variables).

**9.13:** From $\vdash \phi y$, we may pass to $\vdash (x)(\phi x)$, when the latter assertion is well formed.

**individual:** an individual is an object which is neither a proposition nor a function.

**9.131:** individuals are of the same type; elementary functions taking arguments of the same type are of the same type; the negation of a function is of the same type as the function; $\phi\hat{x} \vee \psi\hat{x}$ is of the same type as its disjuncts; $(y)(\phi(\hat{x}, y)$ and $(z)(\psi(\hat{x}, z)$ are of the same type when $\phi\hat{x}\hat{y}$ and $\psi\hat{x}\hat{y}$ are of the same type; propositions are of the same type [we do not subdivide types of propositions – MRH]; Russell provides that elementary propositions are of the same type, that negation does not change type, and that $(x)(\phi x)$ and $(x)(\psi x)$ are of the same type when $\phi\hat{x}$ and $\psi\hat{x}$ are of the same type.

This is copied so I can examine it. My type system is indicated at the end.

**9.14:** If $\phi x$ is significant, then if $x$ is of the same type as $a$, $\phi a$ is significant.

**9.15:** If for some $a$ there is a proposition $\phi a$, then there is a function $\phi\hat{x}$ and vice versa.

On this we impose an exception: this does not hold if $\phi a$ is a proposition in which $a$ appears free in a function notation $\psi\hat{z}$. – MRH

**multiple variables:** surely 9.14 and 9.15 must extend to pfs of more arguments. Here proper definitions of abstraction and substitution for one or many variables are needed.

**note on type system and pfs:** This is where explicit discussion of forms of pfs and the nature of substitution needs to be added.

**note on taking section 10 approach:** We regard all rules above as applying to quantified propositions; we draw no distinction between elementary and other propositions, and indeed do not distinguish types of propositions in practice. – MRH This does require us to add rules for correct use of propositional variables.

**$\alpha$-conversion:** He notes that $(x)\phi x$ is the same proposition as $(y)\phi y$ after 9.21.

**multiple quantifiers:** $(x, y)(\phi xy)$ is defined as $(x)(y)(\phi xy)$, similarly for more variables.

**formal equivalence with two variables:** $\phi xy \equiv_{x,y} \psi xy$ is defined as $(x, y)(\phi xy \leftrightarrow \psi xy)$. Similarly for more variables.

**10.12:** $\vdash (x)(p \vee \phi x) \rightarrow p \vee (x)(\phi x)$ Is he saying this needs to be primitive in the section 10 approach?

**11.07:** "Whatever argument $x$ may be, $\phi(x, y)$ is true whatever argument $y$ may be" implies "Whatever argument $y$ may be, $\phi(x, y)$ is true whatever argument $x$ may be". Incomprehensible, used to interchange universal quantifiers.

**predicative pf notation:** The notation $\phi!x$ means $\phi x$, supplying the additional information that the function $\phi$ is locally predicative.

**12.1:** $\vdash (\exists f)(\phi x \equiv_x f!x)$

**12.11:** $\vdash ((\exists f)(\phi xy \equiv_{x,y} f!xy)$

**types and orders of functions:** This asserts the type system of the introduction rather than that of section 12. The use of locally predicative makes it even a little more general.

A function always has at least one argument.

A function of the first order is one which involves no variables except individuals, either as apparent variables or arguments.

A function of the $(n+1)$th order is one which has at least one argument or bound variable of order $n$ and contains no argument or bound variable over a type of higher order than $n$.

A locally predicative function is one which is first order or of order $n+1$ with an argument of order $n$.

The type of a function is determined by the types of its arguments and its order (and any function has order).

Any function of any number of arguments is equivalent to a locally predicative function of the same arguments (the axiom of reducibility, for which the one and two variable cases are formally stated above).

The intention of the liberalized type system given here is to get a usable system when reducibility is not used.

**13..01:** $x = y$ is defined as $(\phi)(\phi!x \to \phi!y)$

# 5 Appendix: A Proposed Semantics (this should work in combination with the first section, though it might need to be fine tuned)

This section has a different intention. It describes an actual infinitary term model of what I think is the system of PM. The notion of pf abstraction here is actually that of [**?**] or [**?**]: all free variables in a pf are supposed circumflexed. But it models the same theory because of the way reducibility is forced.

Here is what I think is a complete semantics for PM meeting the specifications in the book.

An individual is an element of an infinite but otherwise unspecified set $I$.

The type of individuals 0 is a type of order 0. If $\tau_1, \ldots, \tau_n$ are types and $m$ is a natural number greater than all orders of types $\tau_i$, $(\tau_1, \ldots, \tau_n)^m$ is a type of order $m$. All types are constructed in this way.

A type $(\tau_1, \ldots, \tau_n)^m$ is predicative iff the type $\tau_i$ are 0 or predicative and the order $m$ is one greater than the maximum of the orders of the $\tau_i$'s.

A concrete constant of type 0 is a pair $(0, i)$ for $i \in I$. A concrete constant of type $(\tau_1, \ldots, \tau_n)^m$, where $m$ is one greater than the maximum of the orders of the $\tau_i$'s, is a triple $((\tau_1, \ldots, \tau_n)^m, l, R)$, where $l$ is taken from a class of labels $L$ and $R$ is a set of $n$-tuples $([x_1], \ldots, [x_n])$ where each $x_i$ is a non-variable of type $\tau_i$ and $[x_i]$ is the equivalence class of $x_i$ under an equivalence relation $\sim_{\tau_i}$ on non-variable type $\tau_i$ notations conditions on which are discussed below. There is no assumption that all $((\tau_1, \ldots, \tau_n)^m, l, R)$ are present in a given version of this interpretation (some relations in the ground model may not exist as predicative relations in the interpretation of PM). All the individual constants will be present.

Note that the collection of concrete constants may be of large transfinite size. In this case the language I describe will be of large transfinite size; the various syntactical constructions can nonetheless be coded in set theory (I will not bother to do this explicitly yet). Further, a countable version of the language of PM can be assigned semantics relative to an interpretation of this kind which can be defined in a standard way.

We provide variables of each type (no polymorphism here).

We define propositional notations.

If $X$ is a variable or concrete constant of type $(\tau_1, \ldots, \tau_n)$ and each $x_i$ is a notation of type $\tau_i$, $X(x_1, \ldots, x_n)$ is an atomic propositional notation. The free variables in $X(x_1, \ldots, x_n)$ are exactly those of $X$ and the $x_i$'s which are variables (**not** variables free in $x_i$'s which are not themselves variables). There are no quantified variables in $X(x_1, \ldots, x_n)$.

If $P$ and $Q$ are propositional notations then $\neg P$ and $P \vee Q$ are propositional notations. The free variables in $\neg P$ are the free variables in $P$. The quantified variables in $\neg P$ are the quantified variables in $P$. The free variables in $P \vee Q$ are the free variables in $P$ and the free variables in $Q$. The quantified variables in $P \vee Q$ are the free variables in $P$ and the free variables in $Q$.

If $P$ is a propositional notation in which the variable $x$ is free, $(x)P$ and $(\exists x)P$ are propositional notations, in which the free variables are those free in $P$ other than $x$. The quantified variables in $(x)P$ are the quantified variables in $P$ plus $x$.

A propositional notation which contains at least one free variable is also a propositional function (pf) notation. If the free variables appearing in the notation, in alphabetical order, are $x_1, \ldots, x_n$ and have types $\tau_1, \ldots, \tau_n$ respectively, then the type of the propositional notation is $(\tau_1, \ldots, \tau_n)^m$, where $m$ is the smallest natural number greater than the orders of all the types $\tau_i$ and all the orders of types of the quantified variables in the notation.

We define substitution into a propositional notation.

Let $\sigma$ be a partial map from variables to notations, with the type of $\sigma(x)$ the same as the type of $x$. $\sigma^*(t)$ denotes the result of applying the substitution $\sigma$ to a notation $t$.

$\sigma'(t) = \sigma(t)$ for $t$ a variable in the domain of $\sigma$, and $t$ for all other terms.

$\sigma_x(t) = \sigma(t)$ for $t$ a variable in the domain of $\sigma$ not equal to $x$, undefined otherwise.

$(x \to t)$ is a function whose only domain element is the variable $x$, mapped to a term $t$ of the same type.

$\sigma^*(X(x_1, \ldots, x_n)) = X(\sigma'(x_1), \ldots, \sigma'(x_n)$ if $X$ is not a variable in the domain of $\sigma$, and otherwise $\chi^*(\sigma(X))$, where $\chi$ maps the variables $y_1, \ldots, y_n$ appearing free in $\sigma(X)$ to $\sigma'(x_1), \ldots, \sigma'(x_n)$, respectively.

$\sigma^*(\neg P) = \neg(\sigma^* P); \sigma^*(P \vee Q) = \sigma^* P \vee \sigma^*(Q)$

$\sigma^*((x)(P)) = (z)(\sigma^*_x((x \to z)^* P)); \sigma^*((\exists x)(P)) = (\exists z)(\sigma^*_x((x \to z)^* P))$, with $z$ the first variable not free or quantified in $P$ or in any element of the range of $\sigma$.

We define the truth values of all propositional notations not containing free variables.

$X(x_1, \ldots, x_n)$ will only fail to contain free variables if $X$ is a concrete constant $(\tau, l, R)$ and no $x_i$ is a variable. Its truth value will be true if $([x_1], \ldots, [x_n]) \in R$ and otherwise false.

$\neg P$ will be assigned as its truth value the negation of the true value assigned to $P$. $P \vee Q$ will be assigned as its truth value the disjunction of the truth values assigned to $P$ and $Q$.

$(x)P$ will be assigned the truth value true iff each $(x \to t)^* P$ is assigned the truth value true. $(\exists x)P$ will be assigned the truth value true iff some $(x \to t)^* P$ ($(x \to t)$ being the function sending $x$ to $t$, a non variable term of the same type as $x$) is assigned the truth value true.

The relation $\sim_0$ is equality on individuals.

If propositional notations $P$ and $Q$ of the same type $\tau$ differ only by an order-preserving substitution of all their free variables, then $P \sim_\tau Q$.

If propositional notations $P$ and $Q$ of the same type $\tau$ satisfy $P \sim_\tau Q$ then any terms $P(t_1, \ldots, t_n)$ and $Q(t_1, \ldots, t_n)$ which are assigned truth values will be assigned the same truth values. If the converse is also true we get an extensional model.

If we provide all possible concrete pf terms, we get a model of the Axiom of Reducibility. We probably want to provide equality relations on all types as concrete pf terms in any model (the relations $\sim_\tau$).

The equivalence classes under $\sigma_\tau$ ($\tau \neq 0$) may be taken as representing the pfs in type $\tau$ though I follow Russell in not saying exactly what a pf is, simply relating it to its values.