A realization of the system of Principia Mathematica

M. Randall Holmes

8/2/2017 2:30 pm

Contents

1 Remarks on completing this paper

It needs a bibliography, and probably a few more references.

The section on evaluation of propositions needs to be written out more formally.

The note on p. 15 needs to be cashed out with an actual reference to theorems. I reproduce the relevant test (including a footnote) exactly here: "Now we get hoist by the petard of substitutional quantification. This formula tells us that for each value of f there is an *actual pf notation* ϕ which does the trick [Russell's discussion makes it clear that he thinks this is what he is saying[1]], and in general (if we think our notations are constructed recursively from some finite stock of primitive notions) this is simply not true. Impredicative type theory simply will not have all its predicates represented in this way. NOTE: a proof or reference to a proof of this mathematical fact should be inserted. I have carefully provided a device to make this possible nonetheless: the only way for this to be true is for there to be lots and lots of primitive elementary functions in every type."

A reference is needed for the low consistency strength of PM without the axiom of reducibility.

[1]see for example p. 167: he says that any function of one argument or two is to be equivalent extensionally to a predicative function, and defines a predicative function as one for which we have notation involving no bound variables, which is more general than saying that the function is elementary but not much more so: the subset of the language we are left with is the set of notations built from elementary functions and variable function applications by operations of propositional logic. It really does seem that Russell is here making an assertion about the resources of his language: this assertion is in fact provably false unless one charitably allows elementary functions of all predicative types as primitives.

2 Introduction

This is a document intended to describe a full implementation of the system of the *Principia Mathematica* of Russell[2] and Whitehead (hereinafter PM) meeting modern standards. PM is a piece of mathematics, and there is really just one structure that PM (with the Axiom of Reducibility) can describe (up to a certain amount of isomorphism) and we will duly describe it. This is not to say that some details of an implementation cannot be filled in in different ways: we will discuss this in places in the development where we see other options as possible. Most references are either to the first volume of PM ([**?**], referenced by page number with no identification) or to Klement, K. "The functions of Russell's no-class theory", *Review of Symbolic Logic*, vol. 3, no. 4 Dec 2010 ([**?**], identified briefly as "Klement"), which we read after completing the first draft of this document and found quite congenial on a technical level, though we draw somewhat different philosophical conclusions. Nothing in the main text is modified by anything in Klement; we did add remarks in footnotes to the text as we read the Klement paper.

All of our work on PM owes a debt to the following paper: Fairouz Kamareddine, Twan Laan, and Rob Nederpelt, "Types in Logic and Mathematics before 1940", *Bull. Symbolic Logic*, Volume 8, Number 2 (2002), 185-245 ([**?**]). It is not so much that we take ideas from them directly as that they called our attention to the fact that the system of PM needed to be formalized in accordance with modern standards. Our technical solutions are not always the same as theirs. We are definitely indebted to their notations and terminology. Our paper [**?**], describing a formalization of the type system of PM and software for type-checking terms of a language based on that of PM based on that formalization, should be mentioned only to say that we think that the formalization of the language of PM and its semantics given there should be taken as considerably farther from the actual language of PM and associated semantics than the formalization given in this paper (and closer to that in [**?**]), though comparison of the two papers might be useful to understand the way we think about the subject.

The mathematical implementation of PM which we will present uses the Axiom of Reducibility (so it proves all the results in all three volumes). It

[2]the author of PM is referred to as "Russell" for convenience. We do know that Whitehead is part of the composite author of the work.

uses the type system described in the introduction to the first edition[3], not the type system described in the body of the first edition (section 12) or worse yet in the introduction to the second edition. It uses section 10 for quantifiers: we really can apply propositional connectives to quantified sentences. These are rational decisions designed to make the mathematics clearer.

There are interesting things to be said about PM without the Axiom of Reducibility, but that is another story.

Russell says over and over that the world of PM is inhabited by individuals, propositions, and propositional functions. It is weird and opaque of him to do this and then to provide us with hardly any examples of notation for propositional functions and no examples at all of notation for their types. But the propositional functions are not a fiction and not eliminable. Every quantifier over classes in the whole work (and there are lots of them) translates to a quantifier over propositional functions via the formulas in section 20[4]. Russell is committed to propositional functions, whatever they are [we will explain something that they could very reasonably be].

We feel entirely free to use modern formulations of concepts where they coincide reasonably closely with Russell's treatment. We certainly feel free to use modern notation and to introduce our own notation where desirable.

For us, propositional functions are pieces of notation in a language.[5] This seems to fit closely with how Russell talks about them. This is a specific design decision. We use the briefer term pf for "propositional function" (here we can directly acknowledge the influence of the paper of Kamareddine et. al.). One could alternatively interpret pfs as functions in the usual mathematical sense, but this would run up against Russell's evident unwillingness to commit to their being extensional, and against issues of use and mention: he does in some places pretty clearly regard pfs as characterized by their notational features.[6] One should not forget that notations are themselves abstract objects, and one should note that I am absolutely not committing

[3]pp. 48-55

[4]proposition 20.07 is unequivocal: every quantifier over classes translates to a quantifier over pfs; there is no reduction of ontological claims here, merely a replacement of the claim that there are classes with the claim that there are pfs (whatever they are).

[5]This is (independently) basically the same as Klement's reduction of pfs to open formulas in a language.

[6]An example of such language is found in the four numbered points at the bottom of p. 167 and their application to the discussion of reducibility

to these notations being finitary or making up a countable set. In fact, under the most natural interpretation the "language" I am talking about turns out to be of a quite large uncountable cardinality.[7]

A pf is obtained from an expression for a proposition by circumflexing one or more variables. A pf is actually a sort of λ-term, but the syntax differs from Church's both in the restriction to abstraction from propositional expressions and in the lack of a head binder. A pf is to be thought of as representing a function of a vector of variables, namely the circumflexed variables in its notation, listed in alphabetical order (so $\hat{a} < \hat{b}$ is a function of (a, b) (the less-than relation) and $\hat{b} < \hat{a}$ is also a function of (a, b) (the greater-than relation!).[8]

We *could* take the bull by the horns and replace pfs with λ-terms with proper head binders, and make things much cleaner. We would then write $(\lambda ab : a < b)$ and $(\lambda ab : b < a)$ instead of $\hat{a} < \hat{b}$, $\hat{b} < \hat{a}$, respectively. But it is fun to work out the details and see that Russell's original concept works.

A pf has a type, which is a vector of types consisting of the types of its circumflexed arguments in alphabetical order, with the additional feature of a numerical order, to be explained. Propositions do have types (with order) and there are propositional variables, but for our formal purposes we never circumflex a propositional variable [and Russell only does this under special circumstances]. The basic types are built by this operation from the type of (relative) individuals. We say relative individuals because the typing scheme is polymorphic, with every type being defined relative to the type of relative individuals, which may be any particular type, whose internal structure is being ignored in the context.

[7]Klement discusses realist and nominalist approaches to pfs: I suppose mine is nominalist in his terms though I do not regard my interpretation as giving much comfort to nominalism. An argument in favor of a realist approach (which I could formulate along similar lines) is that distinct pfs may in fact be equal in the internal sense of PM, and it might be better to think of such pfs as names for the same extra-linguistic object. But it is easier not to do this. For Russell speaking as if a pf denotes something, see p. 40, bottom of the second paragraph, but what he says there and near there is quite vague.

[8]The role of alphabetical order of circumflexed variables in computing values of pfs with several arguments is discussed on p. 200. The need for such a device for evaluating pfs with multiple arguments follows from the lack of head binders in the notation.

3 Propositional and Object Notations

To start making this concrete, we need to define the notations for propositions.

propositional variables: A letter $p, q, r \ldots$ may be used as a propositional variable. These have propositional type, (), which may further be qualified as $()^0$ where 0 is the numerical order of the type , as we will reveal. A propositional variable is a propositional notation. No object variable is free in such a notation.

elementary functions: An expression $f(t_1, \ldots, t_n)$ is a propositional notation, where each t_1 is an object term of type τ_i and f is an elementary propositional function of type $(\tau_1, \ldots, \tau_n)^m$, where the numerical order m of the type is the smallest order greater than the orders of each of the τ_i's. We provide many elementary functions for every type vector (in this we are more generous than Russell, who only provides elementary functions of individuals). This represents a simple atomic predicate of a vector of terms of the types τ_i. The elementary function f here is a **constant** (it cannot be circumflexed or generalized over). The type of $f(t_1, \ldots, t_n)$ is $()^m$. The object variables free in this expression are the object variables free in any of the t_i's.

variable function expressions: An expression $x(t_1, \ldots, t_n)$ is a propositional notation, where the t_i's are object terms of types τ_i and where x is a variable of type $(\tau_1, \ldots, \tau_n)^m$, where m is greater than the order of each τ_i. The type of $x(t_1, \ldots, t_n)$ is $()^m$. The object variables free in this expression are the object variables free in each of the t_i's and x. If x is circumflexed or bound by a quantifier, the order of its type is forced to be the smallest which is greater than the orders of the types of the arguments.[9]

One could also use Russell's notation $x!(t_1, \ldots, t_n)$ to force x to have type of lowest order.[10]

[9] At the bottom of p. 165 Russell discusses the fact that in a general term $f(x)$ or $f(\phi(\hat{x})$ the order of f is uncontrolled, but that in any context where it is bound ("apparent") its order will be minimal.

[10] Russell uses the exclamation point to signal that the pf is a *matrix*, that is, involves no quantification, but we do not regard this as a useful concept. We regard quantifications

propositional connectives: If P and Q are propositional notations, $\neg P$ is a propositional notation of the same type as P and with the same free object variables, and $P \vee Q$ is a propositional notation of the propositional type with order the maximum of the orders of the types of P and Q and with those free object variables found in either P or Q. Notations $P \wedge Q$, $P \rightarrow Q$, $P \leftrightarrow Q$ are defined in the usual ways and type in the same way as conjunctions.

quantification: If P is a propositional notation of type $()^n$ and x is an object variable free in P, $(x)P$ is a propositional notation with type the maximum of n and the successor of the order of the type of x. The free object variables in $(x)P$ are those in P other than x. $(\exists x)P$ is defined in the usual way and types in the same way. It may be that x occurs in P with an argument list of circumflexed variables [indicating an explicit type relative to the type of relative individuals]; nonetheless, only the solitary letter x will appear in the quantifier.[11]

completeness: All propositional notations are constructed using the clauses here and no others.

We define object terms.

names for individuals: To enforce the axiom of infinity, we provide an infinite collection of names for distinct individuals, all of the true type 0. We assume that we have a name for each individual.

object variables, type inference, and typed variables: A letter $x, y, z \ldots$ (and possibly other letters) is an object variable. A letter appearing as an argument (not in a quantifier prefix and not in applied position) is of the type 0 of relative individuals with order 0 (of course the order

---

as problematic only when they raise the order of the proposition. He defines predicative functions as matrices in this sense in the body of the first edition (as on p. 167) but in the introduction, on p. 53, the term "predicative" is defined in the more liberal way that we use here: a pf is predicative iff its type has the minimal order compatible with the types of its arguments. On p. 52 he gives the example of a function $\phi!x$ defined as $(y)\psi(x, y)$, making it clear that at that point he is not requiring that functions marked with the exclamation point be matrices.

[11] Russell discusses the omission of type information from quantifier prefixes on p. 165, in paragraph beginning "When any predicative function. . .occurs as apparent variable. . .

of this type is as relative as the type itself: the type of relative individuals may be a more complex type with a nonzero order if we look more closely). If an object variable letter appears in applied position $x(t_1, \ldots, t_n)$ with term t_i having type τ_i, it has a type $(\tau_i, \ldots, \tau_n)^m$ where the order m of this type is higher than the orders of the τ_i's. If the variable x is circumflexed or bound by a quantifier, m is as small as possible. If all the arguments t_i are circumflexed, the expression $x(t_1, \ldots, t_n)$ is in effect a variable with a type label, and the order of its type is taken to be as small as possible The expressions capable of being circumflexed (which we may called typed variables) are exactly the object variable letters and the expressions $x(\hat{y_1}, \ldots, \hat{y_n})$ where all arguments are circumflexed.[12] The only object variable free in x or $x(\hat{y_1}, \ldots, \hat{y_n})$ is x. Note that in a quantified sentence $(x)P$ or $(\exists x)(P)$ we can determine the type of x because x must occur free in P, and this appearance will reveal its type.

pf notations: A pf notation P is not a typed variable and is obtained by applying a circumflex to one or more free typed variables in a propositional notation P_0 [a typed variable is free in P_0 iff the variable free in it is free in P_0]. Let $(\hat{y_1}, \ldots, \hat{y_n})$ be the vector of circumflexed type variables appearing in P but not in P_0, presented in alphabetical order, with types $(\tau_1, \ldots, \tau_n)$. The type of the notation will be $(\tau_1, \ldots, \tau_n)^m$ where m is the smallest order greater than the orders of each τ_i and greater than or equal to the order of P_0 (which may independently be raised by quantifications). There are no free object variables in a pf notation: we view any object variable appearing uncircumflexed in a pf notation as representing a constant[13], and so of course it cannot be bound or quantified over. [this is a limitation of the notation for pfs

[12]These "typed variables" are ubiquitous in Russell's actual notations for propositional functions. See for example the footnote on p. 49 noting that ϕ and $\phi\hat{z}$ are both function variables with similar meaning, the circumflexed argument serving to signal that the function needs an argument, that is, what its type is. More complex typed variables can be seen in use on p. 52.

[13]In the main text I am following what is found in PM, so I am putting a modest proposal regarding constants here in a footnote. We have constant names for each true individual. We could use the names for elementary functions of the type of the relative individuals as constants of that type (since we actually will assume eventually that there is an elementary function coextensional with every pf – we could assume that there was one indiscernible from each predicative pf as readily) and then require that a pf notation is obtained by circumflexing every typed variable in a pf notation, constants being then

of which Russell does not seem to have been aware, but it admits only one solution; the reason that we cannot allow quantification over such a variable is that we take seriously the idea that where $(x)(\phi(x))$ makes sense, so must $\phi(\hat{x})$[14]].

Such notations *do* occur in PM, but Russell does not explicitly note the difficulties that they entail.[15]

A typed variable looks as if it should be a pf notation, and arguably is one, but we distinguish them as a separate kind of notation because they are regarded as containing free occurrences of a variable and substitutions into them are allowed.

A circumflexed variable not appearing as an argument of a typed variable is bound in the smallest notation containing it which looks like a proposition (circumflexes other than those on arguments of typed variables or appearing in proper subnotations for pfs being ignored) and is either in a position appropriate to an object or is the entire term under consideration. Again, we have subtleties due to the fact that we have no head binders for pf terms: the fact that we abstract pfs only from propositions helps avert any ambiguity.

---

syntactically distinct. On the other hand, for purposes of formal reasoning with quantifiers it is convenient to have nonce constants (free variables which cannot be bound) and these could be allowed to occur in pf notations–in fact we would need them to so occur.

[14]proposition 9.15, p. 133. We cannot regard any proposition $f(\hat{x} = a)$ as a function of the variable a for a purely typographical reason: $f(\hat{x} = \hat{a})$ cannot be read as a pf which when applied to a gives $f(\hat{x} = a)$: it *must* be read as the application of f to the pf constant $\hat{x} = \hat{a}$ of two variables, the equality relation on individuals. So we determine that the expression $f(\hat{x} = a)$ cannot be viewed as having a variable a in it: a must be understood as having been fixed as a constant at the time the pf was constructed. This is an example of a problem caused by lack of head binders: λ-notation would allow us to distingush $(\lambda a)(f((\lambda x)(x = a)))$ from $f((\lambda a x)(x = a))$

[15]For example, the notation $\hat{x} = a$ occurs on p. 19. On p. 19 it is also valuable to quote this sentence "In fact, we have found it convenient and possible – except in the explanatory portions–to keep the explicit uses of symbols of the type $\phi\hat{x}$ either as constants (e.g. $\hat{x} = a$) or as real variables, almost entirely out of this work". It is important to note the "almost" and the "explicit": he is not saying that he can dispense with propositional functions or propositional function notation completely, merely that he can keep it to a minimum. The contexual definitions in section 20 of statements about classes, if expanded, will lead to language actually mentioning propositional functions. The secret to his avoiding pf notation is that he never actually expands these notations. But the commitment to pfs remains: the entire work depends on them, as he clearly says in many places.

completeness: All object terms are constructed using the clauses here and no others.

note about circumflexed variables: The expression $\hat{x}(\hat{y_1}, \ldots, \hat{y_n})$ is quite different from $\widehat{x(\hat{y_1}, \ldots, \hat{y_n})}$. There is an error in PM where one is written in the place of the other.[16] The first is a pf notation for a function of $n+1$ variables; the second is a solitary circumflexed variable of a possibly quite complex type (not meaningful except in a larger context).

further note: I have achieved the condition which Russell achieves in PM, where I can type an applied real variable [a variable not circumflexed or quantified over] in a non-predicative way (with order higher than expected) but the type of any circumflexed variable or variable bound by a quantifier (and so any proper constituent of the type of a pf) will be predicative.

All features of this notation are actually found in the text of PM somewhere. This notation is adequate to underpin all the formalism, and it squares with what he says about it.

[16] At the bottom of p. 52 I believe there is a notational error: $f!(\phi!\hat{z})$ is indeed a function of two variables, which should be written $f!(\widehat{\phi!\hat{z}})$ and $\phi!(\hat{z})$: the notation $f!(\hat{\phi}!\hat{z})$ which is actually given can only refer to the application of f to the function $\hat{\phi}!(\hat{z})$, the two argument function which applies a pf to a (relative) individual. This last two argument function is discussed at the top of p. 52, though the circumflexed notation for it is not given, only its ambiguous values $\phi!x$.

4 Evaluation of pfs; the definition of substitution

To complete this definition, we should explain the underlying notion of evaluation of a pf. In particular, the really subtle point (which Russell *does* discuss) is that when one makes a substitution for $\hat{\phi}$ in a pf $\hat{\phi}(t_1, , \ldots, t_n)$, what one obtains is the result of substituting $t_1, \ldots, t_n$ for the appropriate variables in the pf replacing ϕ: there is no intermediate stage at which a pf notation appears in applied position.[17]

Notice that the notation here coincides with the notation for pfs and propositions actually found in PM; the notation for types is a novelty, as PM has no such notation. This follows [I believe] Ramsey ([**?**]). The use of argument lists of circumflexed items as type labels *is* characteristic of PM.

The general convention is that if one asserts a proposition in which some variable is not quantified or circumflexed, this assertion is made for all possible values and all possible typings of these real variables. This allows us to state the axiom of reducibility and some other things of interest exactly as Russell intends. The one point of typing which is not as Russell intends (though he *does* exhibit notations in passing which would exhibit this problem) is the fact that non-circumflexed variables inside pf notations are forced to be constants (which *might* be construed as real variables of global scope): they cannot be bound or circumflexed subsequently.

The formal definition of substitution follows.

[17] this is related to Russell's point at the bottom of p. 39 that there is no presupposition in the form of a sentence ϕx of that particular function ϕ. A pf notation, as opposed to a pf variable, never appears in applied position, because ϕa means exactly the result of replacement of $\hat{x}$ with a in $\phi\hat{x}$. Russell also says on p. 55 that a function is not a constituent of its values: I believe this is what he means.

What we define is $N[t_1/v_1, \ldots, t_n/v_n]$, the result of replacing object variables v_i with object terms t_i simultaneously in N.

propositional variables: $p[t_1/v_1, \ldots, t_n/v_n] = p$.

elementary functions:

$$f(u_1, \ldots, u_m)[t_1/v_1, \ldots, t_n/v_n]$$

$$= f(u_1[t_1/v_1, \ldots, t_n/v_n], \ldots, u_m[t_1/v_1, \ldots, t_n/v_n]).$$

variable functions:

$$x(u_1, \ldots, u_m)[t_1/v_1, \ldots, t_n/v_n]$$

$$= x(u_1[t_1/v_1, \ldots, t_n/v_n], \ldots, u_m[t_1/v_1, \ldots, t_n/v_n]),$$

unless some $v_j = x$, in which case we examine t_j: if t_j is an object variable y,

$$x(u_1, \ldots, u_m)[t_1/v_1, \ldots, t_n/v_n]$$

$$= y(u_1[t_1/v_1, \ldots, t_n/v_n], \ldots, u_m[t_1/v_1, \ldots, t_n/v_n]),$$

but if t_j is a pf with vector of circumflexed variables $(\hat{w_1}, \ldots, \hat{w_m})$ [there must be exactly m variables for this to be well-formed]

$$x(u_1, \ldots, u_m)[t_1/v_1, \ldots, t_n/v_n]$$

$$= t'_j[u_1[t_1/v_1, \ldots, t_n/v_n]/w_1, \ldots, u_m[t_1/v_1, \ldots, t_n/v_n]/w_m],$$

where t'_j is the result of removing circumflexes from the variables circumflexed in forming t_j (not from circumflexed arguments serving to provide typed variables with type signatures). The pf t_j should contain no occurrence of any variable v_i in the substitution.

propositional connectives:

$$(\neg P)[t_1/v_1, \ldots, t_n/v_n] = \neg(P[t_1/v_1, \ldots, t_n/v_n]);$$

$$(P \vee Q)[t_1/v_1, \ldots, t_n/v_n] = P[t_1/v_1, \ldots, t_n/v_n] \vee Q[t_1/v_1, \ldots, t_n/v_n].$$

quantifiers:

$$((x)P)[t_1/v_1, \ldots, t_n/v_n] = (z)(P[z/x][t_1/v_1, \ldots, t_n/v_n])$$

where z is any variable of the same type as x and not occurring in P at all, not identical with any of the v_i's, and not occurring in any of the t_i's at all. We note further that we regard $(x)P$ as exactly equivalent to any $(z)(P[z/x])$ where z does not occur in P at all.

object variables and typed variables:

$$x[t_1/v_1, \ldots, t_n/v_n]$$

is t_i if some v_i is x and otherwise is x. $x(\hat{y_1}, \ldots, \hat{y_m})[t_1/v_1, \ldots, t_n/v_n]$ is $x(\hat{y_1}, \ldots, \hat{y_m})$ if no v_i is x. If v_i is x, this is t_i (or $t_i(\hat{y_1}, \ldots, \hat{y_m})$ if t_i is merely a variable letter).

pf notations: pf notations are constants and will not be affected by substitutions. Non-circumflexed and non-quantified variables appearing in pf notations should not appear as v_i's.

One can prove by induction on the structure of types that this process of substitution preserves types and terminates. This would not be true if type discipline were not applied: replacing x with the purported pf $\neg\hat{x}(\hat{x})$ in $\neg x(x)$ would be a nonterminating substitution, reproducing Russell's paradox.

Notice that polymorphism enters in here: there is a freedom of typing of the t_i's due to the unspecified character of the type of relative individuals. Some t_i may of course force more information about this type, and the type information about all t_j's must then be compatible.

This definition of substitution tells us how to evaluate pfs. We believe that Russell had to avoid explicit use of pf notation because explicit use of it would have forced a formal definition of substitution of this kind, which he did not know how to produce.[18]

[18] on p. 644 of Klement he describes attempts of Russell to develop systematic notation for formal substitutions.

5 Evaluation of propositions: the semantics of (propositional notations of) PM

NOTE: this should be spelled out with more formality.

To complete the semantics, we need to know how to assign truth values to each proposition (evaluation of pfs is handled via the substitution operation: we do not need to assign referents to them). We presume an environment in which values are assigned to each variable which occurs free.

propositional variables: Each propositional variable can be assigned the value true or false in the environment.

elementary functions: Each elementary function f is implemented as a relation on objects of the appropriate types. The relation assigned to f cannot be deduced from syntax: this is background information.

variable functions: The value to be assigned to a variable function is computed by taking the pf value for the applied variable from the environment and carrying out the substitution of the arguments to get a presumably simpler term to evaluate. Substitution does provably simplify terms, so this will actually work.

propositional connectives: We know how to evaluate negations and conjunctions.

quantifications: We can evaluate $(x)P$ as an infinite conjunction of all replacements of x with elements of its type (this is how variables get assigned values in the environment), because all these propositions will be of lower order; substitutional quantification works!

So we have reduced everything to syntax you say. Not really.

5.1 Obligations incurred by the Axiom of Reducibility

The reason has to do with the Axiom of Reducibility, which says

$$(\exists\phi)(x)(f(x) \leftrightarrow \phi(x))$$

where f is a real variable (and so may stand for any pf with type of any order; our typing conventions tell us that ϕ has type of minimum possible order (we do not need an exclamation point on it: just the fact that it is quantified over is enough, because of the way we have defined the typing scheme).

Versions of this axiom are needed for vectors of greater length, of course: Russell provides

$$(\exists\phi)(x)(y)(f(x,y) \leftrightarrow \phi(x,y))$$

Now we get hoist by the petard of substitutional quantification. This formula tells us that for each value of f there is an *actual pf notation* ϕ which does the trick [Russell's discussion makes it clear that he thinks this is what he is saying[19]], and in general (if we think our notations are constructed recursively from some finite stock of primitive notions) this is simply not true. Impredicative type theory simply will not have all its predicates represented in this way. NOTE: a proof or reference to a proof of this mathematical fact should be inserted. I have carefully provided a device to make this possible nonetheless: the only way for this to be true is for there to be lots and lots of primitive elementary functions in every type.[20]

The intuitive way to get Reducibility to work is to provide an elementary function implementing every subset of $\tau_1 \times \ldots \times \tau_n$ where the τ_i's are extensions of types. If done with care, this is a well-founded process, because the elementary function will be of a type of higher order than any of the τ_i's. One can be more subtle (and get the axiom of extensionality to hold in your implementation of PM) by requiring that elementary functions implement all and only the extensions which respect extensionality: a relation R will be implemented iff $R(x_1,\ldots,x_n) \leftrightarrow R(y_1,\ldots,y_n)$ whenever PM asserts that $x_i = y_i$ for each i. The equality of PM on individuals may be supposed to

[19]see for example p. 167: he says that any function of one argument or two is to be equivalent extensionally to a predicative function, and defines a predicative function as one for which we have notation involving no bound variables, which is more general than saying that the function is elementary but not much more so: the subset of the language we are left with is the set of notations built from elementary functions and variable function applications by operations of propositional logic. It really does seem that Russell is here making an assertion about the resources of his language: this assertion is in fact provably false unless one charitably allows elementary functions of all predicative types as primitives.

[20]Klement p. 657 quotes Russell on the possibility of having infinitely (even nondenumerably) many simple symbols

be real equality: on all other types, coextensional pfs of the same type will turn out to be equal. Even the radical approach of allowing all relations on notations to be implemented by elementary functions will work (though it will give a version of PM very far from satisfying extensionality). We cannot require that each pf be implemented by a different elementary function; this would run afoul of Cantor's theorem and it would break the well-foundness of the process [at the point where the elementary functions on a given Cartesian product of types are being constructed, we do not have access to all the impredicative pfs which may have extensions in that type].

In this way we obtain substitutional quantification, but it does nothing to serve what might be thought of as nominalist aims. The ontological commitment to arbitrary sets and relations appears in a quite different place in the semantics, not to do with quantification at all.

5.2 Does the no-class theory reduce ontological commitments?

The claim that the no-class theory effects any reduction of ontological commitments is seriously doubtful. In the translation following number 20, every quantifier over classes [and there are plenty in PM] converts to a quantifier over pfs [this is explicit in 20.07], so one has the commitment to pfs covering the same ground as the commitment to classes, and I have given strong reasons to suppose that pfs commit you to a strong ontology of sets or the equivalent in the presence of Reducibility.[21]

[21] you might argue that substitutional quantification (as opposed to objectual quantification) does not commit you to the existence of the objects of the domain over which you are quantifying – but it does when the domain quantified over is identical with the domain of notations to be substituted, as is the case here. It can be noted that one is not really mentioning but using the notations (p. 651 Klement) but if they are taken to denote anything (including themselves) those things are each being mentioned, or at least each of them is being treated.

More generally, I am skeptical of any application of substitutional quantification to eliminate objects. If you are willing to make your language large enough to provide names for all objects of a class, and willing to provide a non-effective definition of how to evaluate propositions with these names in any context, then you can eliminate *any* domain of objects from explicit consideration. But then you have names for all of these objects and criteria for determining whether they have the same referent (using the Leibniz criterion). Whether you admit it or not, you have the entire domain of objects available to you.

What is really going on here is that the no-class theory is not actually about ontological reduction at all: it is about *extensionality*. Russell is (rather kludgily) presenting a way to interpret an extensional theory in a nonextensional theory. The ontological commitments of the extensional theory (if it were to be taken literally) are of exactly the same strength as those of the nonextensional theory.

It should be noted that there is no commitment in our formalization of PM to a nonextensional view: nothing forces us to provide any predicate which does not treat pfs with the same extension identically, and if we restrict ourselves to such predicates we will find that the theory says that pfs with the same extension (even if different as pieces of infinitary text) are equal: one must remember that equality is a defined notion in PM and look at how it is defined.[22][23]

The Axiom of Reducibility forces us to associate with most complex abstractions of an impredicative character new unanalyzable elementary functions. This is a very strong commitment to the existence of objects specified by abstraction: PM has a considerable commitment to "abstract particulars". Prior to this, one can note that pieces of notation are abstract particulars. Even the version of PM without reducibility must be construed as talking about infinite domains of abstract objects, even if we construe the pfs as PM's own formal notations.

Subsitutional quantification does have value, as in this case, for simplifying the semantics of quantification (at the cost of increasing the size of your language). A claim that a class of objects has been eliminated can be taken more seriously if the class of names is finite or countable and all operations on the class of names that are used are effective.

[22]p. 166: "There is no advantage in assuming that there really are such things as classes...It would seem that the sole purpose which classes serve, and one main reason which makes them linguistically convenient, is that they provide a method of reducing the order of a propositional function. We shall, therefore, not assume anything of what may seem to be involved in the common-sense admission of classes, except this, that every propositional function is equivalent, for all its values, to some predicative function of the same argument or arguments." Our critique is that in fact this grants everything that the assumption of classes permits. Russell seems to think that the paradoxes have something to do with asserting the identity criteria for classes; in fact, imposing extensionality has no logical strength, and the mere ability to talk about nonextensional pfs without type discipline would give the paradoxes at once [the pf $R = \neg\hat{\phi}(\widehat{\phi(\hat{x})}$ would do the trick: the assertion $R(R)$ would be equivalent to $\neg(R(R))$].

[23]Klement p. 642 tells us that Russell knew the paradox cited in the previous footnote.

6 Effects of our design decisions

We briefly comment on our design decisions. The more fine-grained type systems described in the body of the first edition or in the introduction to the second edition make it basically impossible to do any mathematics without the axiom of reducibility. The system of types and orders in the introduction to the first edition is the correct one: it does exactly enough to ensure that substitutional quantification is workable. Russell is excessively cautious, because he is uncertain what exactly is causing the paradoxes; we are more confident now because we have more experience. The approach of section 9 which doesn't allow propositional connectives to be applied to quantified sentences does not gain us anything. We suspect that Russell's motivation in doing this had to do with paranoia about paradoxical assertions, and again, we know [and in this case we think Russell actually knew] that the section 10 approach is perfectly safe. That pfs are actually identified with pieces of notation is perhaps conservative, but workable, noting that the equality of PM is defined, and does not need to be the same as the equality relation on pieces of notation considered in the metatheory. In many places Russell does appear to regard features of the notation associated with a pf to be features of the pf itself.[24]

The one option which we do not take which we do regard as having genuine mathematical interest is the option of denying the Axiom of Reducibility. If one does not assert Infinity one gets a very weak theory indeed. If one has a finite supply of names for individuals, there is a natural model of PM as described above [except for the qualification about the names for individuals] in which each type is finite. Provide an elementary equality relation which respects extensionality on each type and otherwise provide no elementary functions: the resulting interpretation of PM along our lines will think that each type is finite. Further, oddly enough, it will satisfy reducibility! If one

[24]we do note (this is analogous to points discussed by Klement on p. 660) that in fact we have not assigned referents to *any* object symbol in my formal semantics: I have only assigned referents (true or false) to notations for propositions. But I seem to be forced on the metalinguistic level to talk about arbitrary collections of notations in defining the semantics of elementary function symbols. And the linguistic objects which I am using (even if not mentioning) are not concrete linguistic objects (I say things about them which far outrun the scope of any talk of concrete inscriptions). In *some* sense I have described a semantics for the language which does not mention any referents except the True and the False. But the ontological commitments behind this discourse are vast.

does assert Infinity one gets a theory known to be weaker than Peano arithmetic but nonetheless capable of doing a lot of elementary mathematics (it is known that this theory cannot prove mathematical induction or the least upper bound property in analysis). Again, build a minimal model of this by providing only the fixed infinite collection of individuals and the elementary equality relations respecting extensionality on each type. An inductive argument which Russell himself seems to have grasped on an intuitive level shows that the resulting interpretation of PM without reducibility is extensional. In this model Reducibility will be false. One could have fun by doing weird things like providing an elementary enumeration of each type; this would falsify Cantor's theorem. In any models one builds by simple recursions the axiom of reducibility will turn out to be false.

7 Technical issues

7.1 Theory and metatheory

We turn to the matter of how to understand the linguistic expressions that we have used as a substrate for the pfs. We can regard these notations as finite sequences of symbols, or better as finite parse trees, which we can build, all in the same low type, using the machinery of PM itself, from a supply of atomic notations which may be very large[25]. If we wanted to explicitly express quantifications as infinitary conjunctions and disjunctions, this would remove the finitary character of the parse trees, but could still be carried out in a single low type.[26] This is not paradoxical! For PM will view the equality of these internal versions of the propositional notations as genuine textual equality. The relation on propositional notations of having the same truth value is in fact not definable in PM at all (actually, if one considers notations with types of subexpressions bounded as simpler than a given type, it is definable, but only in a type higher than the given type, and this process cannot be described uniformly with the resources of PM). So, although the notations used to implement the propositions and pfs are visible to PM itself, and in fact all visible in the same type, their relation to the semantics of PM is not visible. It is even possible for the notations to make up a countable set (though this is not the natural way to do things): the indefinability of the equivalence of truth value of propositional notations averts paradox.

7.2 The schemes of contextual definition in PM reviewed and found to be eliminable

We recall that $x = y$ is defined as $(\forall \phi)(\phi(x) \leftrightarrow \phi(y))$.

7.2.1 Definite description discussed

We review Russell's contextual definitions. The definite description operator $(\theta x)(P)$ (where x must occur free in P) [we use a different symbol because

[25] If the natural approach is taken, there will be $\beth_\omega$ elementary function symbols

[26] Klement pp. 657-58 on possibility of linguistic expressions of infinite length considered by Russell

we do not know how to typeset his original] is defined by

$$f((\theta x)(P)) \leftrightarrow (\exists a.(\forall x.P \leftrightarrow x = a) \wedge f(a))$$

This can be improved (allowing the logic of expressions $(\theta x)(P)$ to be the same as those of any other variables or constants) by the refinement

$$f((\theta x)(P)) \leftrightarrow (\exists a.(\forall x.P \leftrightarrow x = a) \wedge f(a)) \vee (\neg(\exists a.(\forall x.P \leftrightarrow x = a)) \wedge f(\Delta))$$

where Δ is a default object of the appropriate type: this can be arbitrary in the type of individuals and the universally false pf in each other type.

The use of this approach allows us to regard $(\theta x)(P)$ as in every case the name of a specific object, and removes the need for any considerations of scope in the use of this notation. It must be further observed that the contextual definition can be entirely eliminated (this is basically Frege's approach in [**?**]). Introduce θ as a primitive operation acting on pfs of one variable. The governing axiom would be

$$((\exists a.(\forall x.\phi(x) \leftrightarrow x = a) \wedge \theta(\phi(\hat{x})) = a))$$
$$\vee(\neg(\exists a.(\forall x.\phi(x) \leftrightarrow x = a)) \wedge \theta(\phi(\hat{x})) = \Delta)$$

It is worth noting that the object picked out by a definite description is always of a predicative type, as its type is the type of an argument of a pf. We are not proposing adoption of such an operation, merely pointing out its formal possibility.

7.2.2 The no-class theory discussed

The definition of class notation is thus:

$$f(\hat{x}(P)) = (\exists \phi.(x)(\phi(x) \leftrightarrow P) \wedge f(\phi(\hat{x})))$$

Similarly

$$f(\hat{x}\hat{y}(P)) = (\exists \phi.(x)(y)(\phi(x,y) \leftrightarrow P) \wedge f(\phi(\hat{x},\hat{y})))$$

defines relation notation.

The scope convention for this notation is that in all cases the scope is to be taken to be the smallest proposition containing the class symbol. The idea is that each elementary function symbol or variable function symbol is reinterpreted as having its extension fattened up in the minimal way which causes it to respect coextensionality. This has no effect on the meaning of any symbol which already respects coextensionality.

We propose a uniform method of handling any class variable α: each such variable has a predicative pf type, and is to be read as $\hat{x}\phi_\alpha(x)$ wherever it occurs where $\phi_\alpha\hat{x}$ is a correlated pf variable of the same type. Each quantifier (α) is simply to be read as (ϕ_α). Similarly, each relation variable R is correlated with a pf variable $\phi_R(\hat{x},\hat{y})$ and to be read as $\hat{x}\hat{y}\phi_R(x,y)$ wherever it occurs, with (R) read simply as (ϕ_R). The notation $x \in \alpha$ abbreviates $\phi_\alpha(x)$. The notation $x\,R\,y$ abbreviates $\phi_R(x,y)$. Notice that applied position is a context which respects extension.

7.2.3 The effect of the no-class theory (indeed, a fully extensional theory) achieved with no contextual definition of class at all

It is worth noting that the exact effect of replacing every pf variable with a class variable and so causing extensionality to hold can be achieved in a quite different way (though on examination it is very similar). Define a relation $\sim$ and a new pf application operation $\phi[x_1,\ldots,x_n]$ by mutual recursion. $a \sim b$ for a, b individuals is defined as $a = b$ (using the predicative equality relation on individuals which Reducibility gives us). $\phi \sim \psi$ is defined as

$$(x_1 \ldots x_n, x_1' \ldots x_n')(x_1 \sim x_n \wedge \ldots \wedge x_1' \sim x_n' \rightarrow \phi[x_1,\ldots,x_n] \leftrightarrow \psi[x_1,\ldots,x_n]).$$

$\phi[x_1,\ldots.x_n]$ is defined as $(\exists x_1,\ldots,x_n)((x_1 \sim x_1' \wedge \ldots \wedge x_n \sim x_n' \wedge \phi(x_1,\ldots,x_n))$. The theory of pfs and individuals using the new application operation is extensional and otherwise satisfies the same logic as the original language of pfs, which can then be ignored. No contextual definitions are involved. The relation $\sim$ will coincide with the defined equality of PM in the new language. Also note that order is not raised by replacing all applications with the new application.

One then defines $\hat{x}(\phi(x))$ as $(\theta\psi)((x)(\psi(x) \leftrightarrow \phi(x)))$ [with the contextual definition of θ expressed in terms of the new equality based on $\sim$]. The point

of this is that the type of a definite description is always predicative, due to our typing rules for quantifiers. The unique predicative function with the same extension as ϕ [unique in the sense of the new equality defined as $\sim$] does exist by Reducibility combined with extensionality.

7.2.4 Definite description defined in terms of class notation in almost all cases

Even more dramatically, θ is definable in most types as soon as one has extensionality. $\theta(\phi(\hat{x}))$ can be defined as the class

$$\hat{y}((\exists x.x(y) \wedge (\forall z.\phi(z) \leftrightarrow z = x)))$$

unless the type of x is the type of true individuals. So in fact our device for eliminating the need for contextual definition of classes also eliminates the need for contextual definition of definite description except in the case of its use in defining classes and the case of descriptions of true individuals, and arguably we could do entirely without descriptions of true individuals, as mathematical objects are generally implemented as pfs.

This elimination of most contextual definition from PM is not merely a piece of anachronistic technical brilliance: it should make a point about the actual significance of such devices in the system. In particular it should help to make our point that the no-class theory has nothing to do with ontological reduction at all.

At this point we are fully set up to carry out the actual development in PM in a manner supported by the notation and semantics given here.