A formalization of the ramified type theory

Eindhoven University of Technology

Department of Mathematics and Computing Science

A Formalization of the Ramified Type Theory

by

Twan Laan

94/33

ISSN 0926-4515

editors: prof.dr. J.C.M. Baeten
prof.dr. M. Rem

Computing Science Report 94/33
Eindhoven, August 1994

A formalization of the Ramified Type Theory

Twan Laan
Co-operation Centre Tilburg and Eindhoven Universities
Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O.Box 513, 5600 MB EINDHOVEN, THE NETHERLANDS
e-mail laan@win.tue.nl

September 1, 1994

Abstract

In "Principia Mathematica" [16], B. Russell and A.N. Whitehead propose a type system for higher order logic. This system has become known under the name "ramified type theory". It was invented to avoid the paradoxes, which could be conducted from Frege's "Begriffschrift" [6].
We give a formalization of the ramified type theory as described in the Principia Mathematica, trying to keep it as close as possible to the ideas of the Principia.
As an alternative, distancing ourselves from the Principia, we express notions from the ramified type theory in a lambda calculus style, thus clarifying the type system of Russell and Whitehead in a contemporary setting.
Both formalizations are inspired by current developments in research on type theory and typed lambda calculus; see e.g. [2]. In these formalizations, and also when defining "truth", we will need the notion of ***substitution***. As substitution is not formally defined in the Principia, we have to define it ourselves. Finally, the reaction by Hilbert and Ackermann in [8] on the ramified theory of types, resulting in the "simple type theory", is described.

Contents

1 Introduction

In 1879, Gottlob Frege published the "Begriffschrift" ([6]), in which, for the first time in history, logic was put into a formal system. Unfortunately for Frege, Bertrand Russell proved in 1902 that the system of the "Begriffschrift" is inconsistent.[1] Russell's proof has become known under the name "Russell Paradox".
Though this paradox was not the first one and certainly not the only one in the history of logic and mathematics ([3], Chapter 17, presents a list of paradoxes), Russell's paradox is regarded as the most fundamental one (for instance in [5], pp. 1-7): it is a paradox that arises at the most elementary level of logic.

A solution to this paradox had to be found, and one of these solutions was proposed by Russell himself: the theory of (ramified) types.
Russell remarks[2] that *"An analysis of the paradoxes to be avoided shows that they all result from a certain kind of vicious circle. The vicious circles in question arise from supposing that a collection of objects may contain members which can only be defined by means of the collection as a whole."*.
He avoids this kind of collections by postulating the "vicious circle principle":

Vicious Circle Principle 1.1 *Whatever involves* all *of a collection must not be one of the collection*

With regard to propositions and propositional functions this vicious circle principle results in the Theory of Ramified Types[3].
Russell shows that the Russell Paradox, and some other paradoxes, can not be derived when one accepts the vicious circle principle and the theory of ramified types[4].

The Theory of Types was introduced in [11] (1908) and in the famous "Principia Mathematica" ([16], 1910-1912). In the "Principia", Russell and Whitehead try to base the whole of mathematics on logic. The result is a very formal and accurate build-up of mathematics.
But the fundamentals, among which the theory of types, are not presented that formally. There is no formal definition of type, and no formal definition of proposition or "logic formula".
This may look strange to the modern reader but it is understandable when we project ourselves into the situation at the beginning of this century.
In those days, the notion of formal system was not yet sufficiently developed. Moreover, type theory was not an independent subject as it is today. The theory *"only recommended itself to us in the first instance by its ability to solve certain contradictions"*[5]. And though *"it has also a certain consonance with common sense which makes it inherently credible"*[6], the theory of types was not introduced because it was interesting on its own, but because it had to serve as a tool for logic and mathematics.
This illustrates why there was no need for a formalization of the theory of ramified types in those days.

This paper presents a formalization of the type theory of the Principia, in order to make a link between the type theory of Russell and current type systems.

On the one hand we try to keep our formalization as close as possible to the ideas of the Principia. On the other hand we make the formalization of the ramified theory of types because we want to make a link between the system of the Principia and current type systems in λ-calculus, as the system of the Principia was the first type system.

Our first aim, a formalization close to the Principia, is given in the first part of the paper (sections

[1] Russell discovered the paradox in June 1901, and wrote Frege about it in June 1902. An English translation of this letter can be found in [13]
[2] [16], Introduction, Chapter II, Section I, p. 37
[3] introduced in [16], Introduction, pp. 38–55
[4] [16], Introduction, Chapter II, Section VIII.
[5] [16], Introduction, Chapter II, p. 37.
[6] [16], Introduction, Chapter II, p. 37.

2-8). It is not always easy to give a motivation for the things that we define in these sections. Many notions are not formally defined in the Principia, especially the notions of "matrix" and "substitution".
In Section 2, so-called "pseudomatrices" are introduced. There is no definition of "pseudomatrix" in the Principia, and our definition is only intended to delimit the "things about which we want to speak". All the "matrices" of the Principia[7] are pseudomatrices (though the notation we use differs slightly from the notation of the Principia), and our goal for the sections 3-8 is to find out which pseudomatrices can be called matrices.
In Section 3 we make a first attempt. We introduce types and give a formal definition of the notion "the pseudomatrix f is of type t" (3.9), and motivate our definition by referring to passages in the "Principia". Definition 3.9 forms the heart of this paper. Pseudomatrices to which a type can be assigned will be called (as in the Principia) *matrices*.
As soon as we try to work with this definition, or if we want to prove certain characteristics of it, we find that we have problems with the rule of "substitution". It appears that we do not know exactly what substitution *is* (as a formal machinery). This is quite problematic, as the "Principia" does not spend many words on it. In Section 4 we try to solve this gap in our knowledge by defining substitution. In doing so, we cannot rely only on the "Principia", we also have to use our own intuition and common sense.
When proving some fundamental properties of our definition of substitution, we would like to use the type system as was introduced in 3.9, because it would simplify proofs considerably (for example the proof of Theorem 6.3). But this system is based on the definition of substitution and its properties. So using this type system in the proofs of section 4 would result in a vicious circle. Section 7 brings together the notions of substitution and type (theorems 7.16 and 7.17), and proves some characteristics of 3.9 that we couldn't prove at the end of Section 3.
The question of which pseudomatrices are matrices (according to the definition given at the end of Section 7) is handled in Section 8. We can make a comparison between this notion of matrix, and the notion of matrix as was defined in the "Principia".

Our second aim, the comparison of the system of the Principia with current type systems in λ-calculus, we purse by introducing (in Section 5 and Section 9) a new, λ-calculus-like, notation for pseudomatrices. It appears that "substitution" has much to do with the ordinary notion of β-reduction in λ-calculus, and that our system 3.9 has much in common with, for example, the type systems from the Barendregt Cube ([2]).
Reflecting on the work we did in the previous sections, and translating the results, we find that we have proven variants of well-known theorems like Strong Normalisation, Church-Rosser property, Free Variable Lemma, Thinning Lemma and Unicity of Types.

In Section 10 we complete the formalization of the Ramified Type Theory by introducing quantifiers. It is characteristic for the system of the Principia, that quantifiers are introduced at the last stage. We mention the reasons for this late introduction, but also arguments against it.

The ramified theory of types is a very restrictive system to develop mathematics in. Whitehead and Russell solved this problem by postulating the "Axiom of Reducibility". When you want to develop mathematics using the ramified type theory, it is very attractive to accept this axiom. But it is almost impossible to give another motivation for this axiom.
Ramsey ([9], 1926) and Hilbert and Ackermann ([8], 1928) propose a simplification of the ramified type theory (the *simple* type theory), in which mathematics is easier to develop. They motivate this simplification by making the distinction between logical and semantical paradoxes. It turns out to be very easy to carry out this simplification in our formal system 3.9.

2 Pseudomatrices

In this section we shall describe the set of propositions and propositional functions which Whitehead and Russell use in the Principia, and which we call "pseudomatrices". We give a modernised, formal

[7] see [16], Introduction, Chapter II, Section V, pp. 50-52, and Part I, Section B, *12, pp.162-164.

definition which corresponds to the description in the Principia.

At the basis of the system of our formalization there is

- a set $\mathcal{A}$ of *individual-symbols*;
- a collection $\mathcal{V}$ of *variables*;
- a set $\mathcal{R}$ of *relation-symbols* together with a map $\mathfrak{a} : \mathcal{R} \to \mathbb{N}$ (indicating the *arity* of each relation-symbol).

As functions are represented as relations in the Principia, we will not introduce a special set of function symbols.
We will use the letters $x, y, z, x_1, \ldots$ as meta-variables over $\mathcal{V}$, and $R, R_1, \ldots$ as meta-variables over $\mathcal{R}$.
We assume that there is an *alphabetical order* on the collection $\mathcal{V}$, and write $x < y$ if the variable x is ordered before the variable y according to this alphabetical order. We assume that

$$\begin{aligned} \{\mathtt{a}_1, \mathtt{a}_2, \ldots\} &\subseteq \mathcal{A} \\ \{\mathtt{x}, \mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{y}, \mathtt{y}_1, \ldots, \mathtt{z}, \mathtt{z}_1, \ldots\} &\subseteq \mathcal{V} \\ \{\mathtt{R}, \mathtt{R}_1, \ldots, \mathtt{S}, \mathtt{S}_1, \ldots\} &\subseteq \mathcal{R}; \end{aligned}$$

moreover:

$$\mathtt{x} < \mathtt{x}_1 < \ldots < \mathtt{y} < \mathtt{y}_1 < \ldots < \mathtt{z} < \mathtt{z}_1 < \ldots$$

and: for each variable $x \in \mathcal{V}$ there is $y \in \mathcal{V}$ with $x < y$.

Definition 2.1 We define a collection $\mathcal{P}$ of *pseudomatrices*, and for each element f of $\mathcal{P}$ we simultaneously define the collection $\textsc{fv}(f)$ of *free variables* of f:

1. If $R \in \mathcal{R}$ and $i_1, \ldots, i_{\mathfrak{a}(R)} \in \mathcal{A} \cup \mathcal{V}$ then $R(i_1, \ldots, i_{\mathfrak{a}(R)}) \in \mathcal{P}$.
   $\textsc{fv}(R(i_1, \ldots, i_{\mathfrak{a}(R)})) \stackrel{\text{def}}{=} \{i_1, \ldots, i_{\mathfrak{a}(R)}\} \cap \mathcal{V}$;
2. If $z \in \mathcal{V}$, $n \in \mathbb{N}$ and $k_1, \ldots, k_n \in \mathcal{A} \cup \mathcal{V} \cup \mathcal{P}$, then $z(k_1, \ldots, k_n) \in \mathcal{P}$ and $\textsc{fv}(z(k_1, \ldots, k_n)) \stackrel{\text{def}}{=} \{z, k_1, \ldots, k_n\} \cap \mathcal{V}$.
   If $n = 0$ then we write $z()$ in order to distinguish the pseudomatrix $z()$ from the variable z[8];
3. If $f, g \in \mathcal{P}$ then $f \vee g \in \mathcal{P}$ and $\neg f \in \mathcal{P}$. $\textsc{fv}(f \vee g) \stackrel{\text{def}}{=} \textsc{fv}(f) \cup \textsc{fv}(g)$; $\textsc{fv}(\neg f) \stackrel{\text{def}}{=} \textsc{fv}(f)$;
4. All pseudomatrices can be constructed by using the construction-rules 1, 2 and 3 above.

We use the letters f, g, h as meta-variables over $\mathcal{P}$.

Remark 2.2 The intuition behind a pseudomatrix is the (naive) idea of *propositional function* as presented in the Principia[9] together with the idea of *proposition*. A propositional function can be seen as an expression with one or more free variables. It will turn into a proposition as soon as we assign values to all the free variables occurring in it. In this light, a *proposition* can be seen as a degenerated propositional function (having 0 free variables).
It will be clear now what the intuition behind pseudomatrices of the form $R(i_1, \ldots, i_{\mathfrak{a}(R)})$, $f \vee g$ and $\neg f$ is. In a pseudomatrix of the form $z(k_1, \ldots, k_n)$ one can see z as a variable for a pseudomatrix with n free variables. $k_1, \ldots, k_n$ indicate what should be substituted for these free variables as soon as an appropriate value is assigned to z.
In the Principia, it is not made clear *how* we should carry out such substitutions. We must depend on our intuition and on the way in which substitution is *used* in the Principia. Unfortunately, there are no really complex examples of the use of substitution in the Principia.

[8] It is important to note that a variable is not a pseudomatrix. See for instance [10], Chapter VIII: "The variable", p.94 of the 7th impression.

[9] See for instance: Introduction, Chapter II, Section II about the nature of propositional functions

Remark 2.3 Note that pseudomatrices do not obey to the Vicious Circle Principle 1.1 yet! $\neg \mathtt{z}(\mathtt{z})$ is, for example, a pseudomatrix. In the next sections we will assign types to some pseudomatrices. The pseudomatrices to which a type can be assigned will be called matrices (Definition 3.10), and it will be shown (Theorem 8.12) that no type can be assigned to the pseudomatrix $\neg \mathtt{z}(\mathtt{z})$.

Remark 2.4 One should be careful with the definition of free variable. At first sight one might think that $\mathrm{FV}(\mathtt{z}_1(\mathtt{x}_1, \mathtt{x}_2, \mathtt{R}(\mathtt{x}_3, \mathtt{x}_4))) = \{\mathtt{x}_1, \mathtt{x}_2, \mathtt{x}_3, \mathtt{x}_4, \mathtt{z}_1\}$.
If we take a closer look at the definition above, it turns out that

$$\mathrm{FV}(\mathtt{z}_1(\mathtt{x}_1, \mathtt{x}_2, \mathtt{R}(\mathtt{x}_3, \mathtt{x}_4))) = \{\mathtt{x}_1, \mathtt{x}_2, \mathtt{z}_1\}.$$

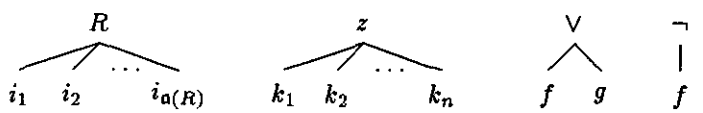
Let's see whether the latter is in agreement with the intuitive interpretation of pseudomatrices given in Remark 2.2, by assigning the value $\mathtt{z}_2(\mathtt{x}_5, \mathtt{x}_6)$ to the variable $\mathtt{z}_1$, $\mathtt{a}_1$ to $\mathtt{x}_1$ and $\mathtt{a}_2$ to $\mathtt{x}_2$. This results in $(\mathtt{z}_2(\mathtt{x}_5, \mathtt{x}_6))(\mathtt{a}_1, \mathtt{a}_2, \mathtt{R}(\mathtt{x}_3, \mathtt{x}_4))$ and our interpretation prescribes that $\mathtt{a}_1, \mathtt{a}_2$ and $\mathtt{R}(\mathtt{x}_3, \mathtt{x}_4)$ should be assigned (*in some order*) to the free variables of $\mathtt{z}_2(\mathtt{x}_5, \mathtt{x}_6)$.
A possibility is to assign the value $\mathtt{R}(\mathtt{x}_3, \mathtt{x}_4)$ to $\mathtt{x}_5$, $\mathtt{a}_1$ to $\mathtt{x}_6$ and $\mathtt{a}_2$ to $\mathtt{z}_2$, but we will assign the values in a more reasonable way: We'll assign $\mathtt{R}(\mathtt{x}_3, \mathtt{x}_4)$ to $\mathtt{z}_2$, $\mathtt{a}_1$ to $\mathtt{x}_5$ and $\mathtt{a}_2$ to $\mathtt{x}_6$ (In the next sections, we will use the types and the alphabetical order on the variables, to decide about which value is assigned to which variable).
This results in $(\mathtt{R}(\mathtt{x}_3, \mathtt{x}_4))(\mathtt{a}_1, \mathtt{a}_2)$, and, again using the intuitive notion of the previous remark, we should assign the values $\mathtt{a}_1, \mathtt{a}_2$ to the free variables $\mathtt{x}_3$ and $\mathtt{x}_4$ of $\mathtt{R}(\mathtt{x}_3, \mathtt{x}_4)$. This, finally, results in either $\mathtt{R}(\mathtt{a}_1, \mathtt{a}_2)$ or $\mathtt{R}(\mathtt{a}_2, \mathtt{a}_1)$, and both are indeed propositions.
It will be clear from this example that it suffices to assign values to $\mathtt{x}_1$, $\mathtt{x}_2$ and $\mathtt{z}_1$, in order to obtain a meaningful result. The variables $\mathtt{x}_3$ and $\mathtt{x}_4$ obtain their values "in the way of the substitution process". That's why these variables are not incorporated in the initial free variable set. In Section 4, we will give a formal definition of substitution, and in Theorem 6.7, we prove that the definitions of substitution and free variable are in line with the intuitive interpretation of Remark 2.2.

We conclude our discussion of pseudomatrices with defining a number of related notions:

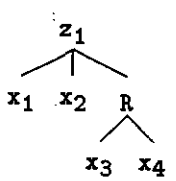
Definition 2.5 Let f and g be pseudomatrices. We say that f and g are *α-equal*, notation $f =_\alpha g$, if there is a bijection $\varphi : \mathcal{V} \to \mathcal{V}$ such that g can be obtained from f by replacing each variable x that occurs in f by $\varphi(x)$.

Definition 2.6 Pseudomatrices of the form $R(i_1, \ldots, i_{a(R)})$ or $z(k_1, \ldots, k_n)$ will be called *simple*; pseudomatrices of the form $f \vee g$ or $\neg f$ will be called *complex*.
The only *submatrix* of a pseudomatrix of the form $R(i_1, \ldots, i_{a(R)})$ is $R(i_1, \ldots, i_{a(R)})$ itself;
The only submatrix of a pseudomatrix of the form $z(k_1, \ldots, k_n)$ is $z(k_1, \ldots, k_n)$ itself;
The submatrices of $f \vee g$ are the submatrices of f, the submatrices of g and $f \vee g$ itself;
The submatrices of $\neg f$ are the submatrices of f and $\neg f$ itself.

Clearly, pseudomatrices can be represented as *trees*. The following pictures speak for themselves:



Example 2.7 *The following tree represents the pseudomatrix* $\mathtt{z}_1(\mathtt{x}_1, \mathtt{x}_2, \mathtt{R}(\mathtt{x}_3, \mathtt{x}_4))$*:*



Definition 2.8 Assume f is a pseudomatrix, and $k \in \mathcal{A} \cup \mathcal{V} \cup \mathcal{P}$. We define, inductively, the notion *k is an argument of f*.

1. If $f = R(i_1, \ldots, i_{a(R)})$ then $i_1, \ldots, i_{a(R)}$ are the arguments of f.
2. If $f = z(k_1, \ldots, k_n)$ then $k_1, \ldots, k_n$ are the arguments of f.
3. If $f = f_1 \vee f_2$ then the arguments of f are the arguments of f_1 together with the arguments of f_2;
   if $f = \neg f'$ then the arguments of f are the arguments of f'.

Example 2.9 *Look at the pseudomatrix* $f \stackrel{\text{def}}{=} \mathtt{z}(\mathtt{x_1}, \mathtt{x_2}, \mathtt{R}(\mathtt{x_1}, \mathtt{x_2}))$.
$\mathtt{x_1}$, $\mathtt{x_2}$ *and* $\mathtt{R}(\mathtt{x_1}, \mathtt{x_2})$ *are the arguments of* f, *but the rightmost occurrence of* $\mathtt{x_1}$ *is* not *an argument of* f.

Remark 2.10 Looking at this example and thinking about λ-calculus, one might think about a "variable-convention", i.e. making the convention that in a pseudomatrix, the free variables differ from all the non-free variables.
Variable conventions are not mentioned in the Principia. However, there is no place where the variable convention is violated. For the time being, we will not make a variable convention. In Section 9, we will compare the system that we are going to introduce with λ-calculus, and there, the variable convention is also discussed (Example 9.5).

We will sometimes need arguments of arguments, and arguments of arguments of arguments. We therefore define:

Definition 2.11 Let $f \in \mathcal{P}$. We define $\mathrm{RA}(f)$, the *recursive arguments* of f:

- If $f = R(i_1, \ldots, i_{a(R)})$ then $\mathrm{RA}(f) = \{i_1, \ldots, i_{a(R)}\}$;
- If $f = z(k_1, \ldots, k_m)$ then $\mathrm{RA}(f) = \{k_1, \ldots, k_m\} \cup \bigcup_{k_i \in \mathcal{P}} \mathrm{RA}(k_i)$;
- If $f = f_1 \vee f_2$ then $\mathrm{RA}(f) = \mathrm{RA}(f_1) \cup \mathrm{RA}(f_2)$;
  if $f = \neg f'$ then $\mathrm{RA}(f) = \mathrm{RA}(f')$.

3 Formalization of Ramified Type Theory

We will introduce the notion of *simple type*.

Definition 3.1 (Simple types)

1. 0 is a simple type;
2. If $t_1, \ldots, t_n$ are simple types, then also $(t_1, \ldots, t_n)$ is a simple type. $n = 0$ is allowed: then we obtain the simple type ().
3. All simple types can be constructed using the rules 1 and 2.

Remark 3.2 (Simple) types are not explicitly introduced in the Principia. But there is a definition of "being of the same type"[10], on which definition 3.1 is based.
In [16], *9·131, is defined, that *"individuals are of the same type"*. This will be the type 0.
The type $(t_1, \ldots, t_n)$ stands for the propositional functions with n arguments ("free variables"), say $x_1, \ldots, x_n$, such that if we assign values k_1 of type t_1 to x_1, k_2 of type t_2 to x_2, ..., k_n of type t_n to x_n, then we obtain a proposition.
In particular, the type () stands for the type of the propositions. Remark, that the order (sequence) of the t_is in $(t_1, \ldots, t_n)$ is not yet essential here.

[10] [16], *9·131, p. 133

Example 3.3 *We give an example to illustrate the intuition presented in Remark 3.2: Look again at the pseudomatrix* $\mathtt{z_1(x_1, x_2, R(x_3, x_4))}$ *(we already spoke about this pseudomatrix in Remark 2.4). In Remark 2.4 we substituted individual symbols for the variables* $\mathtt{x_1, x_2}$*, and during this calculation, also individual symbols are assigned to the variables* $\mathtt{x_3, x_4}$*.*
So the calculation in Remark 2.4 shows, that the pseudomatrix $\mathtt{R(x_3, x_4)}$ *must be of type* $(0,0)$*: When two individuals are substituted, the pseudomatrix becomes a proposition.*
Now we can also decide about the type of pseudomatrices that can be substituted for $\mathtt{z_1}$*: For* $\mathtt{z_1}$ *we can only substitute pseudomatrices with three free variables: Two of these free variables must be of the same type as* $\mathtt{x_1}$ *and* $\mathtt{x_2}$ *(so: type* 0*); the third one must have the type of* $\mathtt{R(x_3, x_4)}$*:* $(0,0)$*.*
That means that for $\mathtt{z_1}$*, only pseudomatrices of type* $(0,0,(0,0))$ *can be substituted.*
Now look at the full pseudomatrix $\mathtt{z_1(x_1, x_2, R(x_3, x_4))}$*. This pseudomatrix has three free variables:* $\mathtt{x_1}$ *and* $\mathtt{x_2}$*, for which only things of type* 0 *can be substituted, and* $\mathtt{z_1}$*, for which only things of type* $(0,0,(0,0))$ *can be substituted.*
Hence, the pseudomatrix $\mathtt{z_1(x_1, x_2, R(x_3, x_4))}$ *must be of type* $(0,0,(0,0,(0,0)))$*.*

The intuition presented in Remark 3.2 and Example 3.3 will be made formal in 3.9. The theorems 7.6 and 7.16 show that this formalization coincides with the intuition.

Simple types can be divided into orders[11]. An order is simply a natural number.
Then we obtain *ramified types*:

Definition 3.4 (Ramified types)

1. 0^0 is a ramified type;
2. If $t_1^{a_1}, \ldots, t_n^{a_n}$ are ramified types, and $a \in \mathbb{N}$, $a > \max(a_1, \ldots, a_n)$, then $(t_1^{a_1}, \ldots, t_n^{a_n})^a$ is a ramified type (if $n = 0$ then take $a \geq 0$);
3. All ramified types can be constructed using the rules 1 and 2.

If t^a is a ramified type, then a is called the *order* of t^a.

Pseudomatrices do not have quantifiers $\forall, \exists$. Orders will become important from section 10 on, where quantifiers are introduced, but until then, their influence is marginal.

Example 3.5 *The following lists of symbols are ramified types:*

- 0^0;
- $(0^0)^1$;
- $((0^0)^1, (0^0)^4)^5$;
- $(0^0, ()^2, (0^0, (0^0)^1)^2)^7$.

But $(0^0, (0^0, (0^0)^2)^2)^7$ *is not a ramified type.*

In the rest of this paper we mainly use ramified types. We simply speak of *types* when we mean: *ramified* types, as long as no confusion arises.

Definition 3.6 (Basic formulas) If $R \in \mathcal{R}$ and $c_1, \ldots, c_{a(R)} \in \mathcal{A}$, then the pseudomatrix $R(c_1, \ldots, c_{a(R)})$ is called a *basic formula*.

The basic formulas are formalizations of the "elementary judgements" in the Principia[12].

Definition 3.7 (Contexts) Let $x_1, \ldots, x_n \in \mathcal{V}$ be distinct variables, and assume $t_1^{a_1}, \ldots, t_n^{a_n}$ are ramified types. Then $\{x_1{:}t_1^{a_1}, \ldots, x_n{:}t_n^{a_n}\}$ is a *context*. $\{x_1, \ldots, x_n\}$ is called the *domain* of the context. Notation: $\mathrm{dom}(\{x_1{:}t_1^{a_1}, \ldots, x_n{:}t_n^{a_n}\})$.

[11] More about orders can be found in [16], *12 and Introduction, Chapter II, Section V, pp. 51-55
[12] [16], Introduction, Chapter II, Section III, pp. 43-45, and Introduction to the 2nd edition, Section I, p. xv.

These contexts will take over implicit presuppositions: In Example 3.3 for instance, we were working under the assumption that for the variable $\mathbf{x_1}$ only individual symbols (things of type 0^0) can be substituted. In the formal system that we are going to build, such information will be made explicit by the addition of contexts.
We will use the letters Γ, Δ for meta-variables over contexts.

Sometimes, the notion of α-equality is too general in contexts. Take for example the context $\{\mathbf{x_1}{:}0^0, \mathbf{x_2}{:}0^0, \mathbf{y}{:}(0^0)^1, \mathbf{z}{:}((0^0)^1)^2\}$. The pseudomatrices $\mathbf{y}(\mathbf{x_1})$ and $\mathbf{y}(\mathbf{x_2})$ as well as the pseudomatrices $\mathbf{y}(\mathbf{x_1})$ and $\mathbf{z}(\mathbf{y})$ are α-equal. But sometimes we want $\mathbf{y}(\mathbf{x_1})$ and $\mathbf{y}(\mathbf{x_2})$ to be "equal" and $\mathbf{y}(\mathbf{x_1})$ and $\mathbf{z}(\mathbf{y})$ not: $\mathbf{x_1}$ and $\mathbf{x_2}$ have the same type in this context, while the types of $\mathbf{y}$ and $\mathbf{z}$ are different, and the types of $\mathbf{x_1}$ and $\mathbf{y}$ are, too.
And if we take the context $\{\mathbf{x}{:}0^0, \mathbf{y}{:}(0^0)^1, \mathbf{z}{:}0^0\}$ and observe the pseudomatrices $\mathbf{y}(\mathbf{x})$ and $\mathbf{y}(\mathbf{z})$, then we see that these pseudomatrices are α-equal, and that the "corresponding" variables have the same type, but their alphabetic order is different: $\mathbf{x} < \mathbf{y}$ but $\mathbf{z} > \mathbf{y}$. Therefore we define the notion of α_Γ-equality:

Definition 3.8 Assume Γ is a context and f and g are pseudomatrices. f and g are called α_Γ-equal, if there is a bijection $\varphi : \mathcal{V} \to \mathcal{V}$ such that g can be obtained from f by replacing each variable x that occurs in f by $\varphi(x)$, and for all x that occur in f: $x : t^a \in \Gamma \Leftrightarrow \varphi(x) : t^a \in \Gamma$, and for all x, y that occur in f: $x < y \Leftrightarrow \varphi(x) < \varphi(y)$.

We will now define what we mean by $\Gamma \vdash f : t^a$, in words: f is of type t^a in the context Γ. In this definition we will try to follow the line of the Principia ([16]) as much as possible. If $\Gamma = \emptyset$ then we will write $\vdash f : t^a$.

Definition 3.9 (Ramified Theory of Types: RTT) We will define the *judgement* $\Gamma \vdash f : t^a$ inductively:

1. (start) For all individual symbols a:
$$\vdash a : 0^0$$
For all basic formulas f:
$$\vdash f : ()^0$$

2. (connectives) Assume $\Gamma \vdash f : (t_1^{a_1}, \ldots, t_n^{a_n})^a$ and $\Delta \vdash g : (u_1^{b_1}, \ldots, u_m^{b_m})^b$, and for all $x \in \mathrm{dom}(\Gamma)$ and $y \in \mathrm{dom}(\Delta)$, $x < y$. Then
$$\Gamma \cup \Delta \vdash f \vee g : (t_1^{a_1}, \ldots, t_n^{a_n}, u_1^{b_1}, \ldots, u_m^{b_m})^{\max(a,b)}$$
$$\Gamma \vdash \neg f : (t_1^{a_1}, \ldots, t_n^{a_n})^a$$

3. (introduction of variables, 1) If $\Gamma \vdash f : (t_1^{a_1}, \ldots, t_m^{a_m})^a$, and $g \in \mathcal{A} \cup \mathcal{P}$ is an argument of f, and $\Gamma \vdash g : t_{m+1}^{a_{m+1}}$, and $y < z$ for all $y \in \mathrm{dom}(\Gamma)$, then
$$\Gamma' \vdash h : (t_1^{a_1}, \ldots, t_{m+1}^{a_{m+1}})^{\max(a, a_{m+1}+1)}$$
Here, h is a pseudo-matrix obtained by replacing all arguments g' of f which are α_Γ-equal to g by z. Γ' is the subset of the context $\Gamma \cup \{z : t_{m+1}^{a_{m+1}}\}$ such that $\mathrm{dom}(\Gamma')$ contains exactly all the variables that occur in h[13].

4. (introduction of variables, 2) If $\Gamma \vdash f : (t_1^{a_1}, \ldots, t_m^{a_m})^a$ and g is a submatrix of f, and $\Gamma \vdash g : t_{m+1}^{a_{m+1}}$, and $y < z$ for all $y \in \mathrm{dom}(\Gamma)$, then
$$\Gamma' \vdash h : (t_1^{a_1}, \ldots, t_{m+1}^{a_{m+1}})^{\max(a, n+1)}.$$
Here, h is the pseudomatrix obtained by replacing all submatrices g' of f which are α_Γ-equal to g by $z(x_1, \ldots, x_p)$ (where $x_1 < \ldots < x_p$ are the free variables of the g' in question).
Γ' is the subset of the context $\Gamma \cup \{z : t_{m+1}^{a_{m+1}}\}$ such that $\mathrm{dom}(\Gamma')$ contains exactly all the variables that occur in h[14].

[13] In Lemma 3.16 we prove that this context always exists.
[14] The existence of this context is showed in Lemma 3.16.

5. (weakening) If Γ, Δ are contexts, and $\Gamma \subseteq \Delta$, and $\Gamma \vdash f : t^m$, then also $\Delta \vdash f : t^m$;

6. (substitution) If y is the ith free variable in f (according to the alphabetic order), and $\Gamma \cup \{y : t_i^{a_i}\} \vdash f : (t_1^{a_1}, \ldots, t_n^{a_n})^a$, and $\Gamma \vdash k : t_i^{a_i}$ then

$$\Gamma' \vdash f[y{:=}k] : (t_1^{a_1}, \ldots, t_{i-1}^{a_{i-1}}, t_{i+1}^{a_{i+1}}, \ldots, t_n^{a_n})^b$$

   Here, $b = 1 + \max(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n)$, ($b = 0$ if $n = 1$) and once more, Γ' is the subset of $\Gamma \cup \{y : t_i^{a_i}\}$ such that $\mathrm{dom}(\Gamma')$ contains exactly all the variables that occur in $f[y{:=}k]$[15]. By $f[y{:=}k]$ we mean the pseudomatrix where k is "substituted" for all free occurrences of y (We come back to substitution in Section 4).

7. (permutation) If y is the ith free variable in f, and $\Gamma \cup \{y{:}t_i^{a_i}\} \vdash f : (t_1^{a_1}, \ldots, t_n^{a_n})^a$, and $x < z$ for all $x \in \mathrm{dom}(\Gamma)$, then

$$\Gamma' \cup \{z{:}t_i^{a_i}\} \vdash f[y{:=}z] : (t_1^{a_1}, \ldots, t_{i-1}^{a_{i-1}}, t_{i+1}^{a_{i+1}}, \ldots, t_n^{a_n}, t_i^{a_i})^a.$$

   $\Gamma' = \Gamma$ if y does not occur in $f[y{:=}z]$; $\Gamma' = \Gamma \cup \{y{:}t_i^{a_i}\}$ if y occurs in $f[y{:=}z]$.

Definition 3.10 A pseudomatrix f is called a *matrix*, if there is a context Γ and a ramified type t^a such that $\Gamma \vdash f : t^a$.

Remark 3.11 We will motivate definition 3.9 by referring to the Principia:

1. Individuals and elementary judgements (basic formulas) are, also in the Principia, the basic ingredients for creating matrices.[16]

2. We only introduce the logical connectives $\vee$ and $\neg$. This is in line with the Principia, where only negation and disjunction are taken as primitive ideas[17], and implication[18] and conjunction[19] are defined in terms of negation and disjunction.
   We can see rule 2 "at work" in *12, p. 163 of the Principia[20]: *"We can build up a number of new matrices, such as* [...] $\varphi!x \vee \varphi!y$, $\varphi!x \vee \psi!x$, $\varphi!x \vee \psi!y$, [...] *and so on."*
   The restriction about contexts that we make in rule 2 is not made in the Principia, and will be discussed in 3.13.

3. Rule 3 is justified by *9·14 and *9·15 in the Principia.

4. Rule 4 looks very much like rule 3, but when we take a closer look then it appears to be completely different from rule 3.
   First of all, the g in rule 3 can not contain any free variables of f (g is an argument of f, so an occurrence of a variable x in g can not be a free variable of f), while the free variables of the g in rule 4 are also free variables of f. That makes clear why in rule 3, g is replaced by just a variable; while in rule 4, g is replaced by the pseudomatrix $z(x_1, \ldots, x_p)$: In rule 4 we don't want to lose the free variables $x_1, \ldots, x_p$ of g.
   Moreover, the idea in the Principia that is behind our rule 4 is not *9·14 and *9·15, but is based on the Introduction, Chapter II, Section V, p. 51. There, matrices are constructed, and *"the first matrices that occure are those whose values are of the forms*

$$\varphi x, \psi(x, y), \chi(x, y, z, \ldots),$$

[15] Again, the existence of Γ' is proved in 3.16.

[16] As for individuals: see [16], *9, p. 132, where "Individual" is presented as a primitive idea. As for elementary judgements: See [16], Introduction, Chapter II, Section III, pp. 43-45.

[17] cf. [16], *1, pp. 93-94

[18] cf. [16], *1·01, p. 94

[19] cf. [16], *3·01, p. 107

[20] In the Principia, Russell and Whitehead write $\varphi!x$ instead of φx to indicate that φx is not only (what we would call) a pseudomatrix, but even a matrix.

i.e. *where the arguments, however many there may be, are all individuals.* [...] *Such functions we will call* 'first-order *functions.'*
We may now introduce a notation to express 'any first-order function.' [...]" This, then, results in second order functions, and this process is repeated to obtain third-order functions and functions of higher orders.
When we apply rule 3, the type of the g that is replaced, and especially its order, was already known in the premises of rule 3. Rule 4, however, makes it possible to introduce variables of higher order.
In fact, leaving out rule 4 would lead to the well-known first-order predicate logic as without rule 4 it is impossible to introduce variables of types that differ from $\mathtt{o}^0$.

5. The weakening rule cannot be found in the Principia, because no formal contexts are used there. It is implicitly present, however, as contexts are implicitly present. And starting to speak about an additional variable does not affect the well-typedness of matrices that were already constructed.

6. The rule of substitution is based on *9·14 and *9·15 of the Principia. We must notice that substitution is not defined in the Principia, so it is not completely clear yet what $f[y{:=}k]$ should mean. Clarification of the notion of substitution is the main topic of Sections 4–6.

7. In Remark 3.2, we observed that the order (sequence) of the t_is in a type $(t_1,\ldots,t_n)$ is not essential in the Principia. More precise: If φ is a bijection $\{1,\ldots,n\}\to\{1,\ldots,n\}$, then the type $(t_1,\ldots,t_n)$ is "the same" type as $(t_{\varphi(1)},\ldots,t_{\varphi(n)})$.
In the system above, the order of the t_is appears to be related to the alphabetic order of the free variables of the pseudomatrix f that has type $(t_1,\ldots,t_n)$ (see Theorem 7.6). We need this alphabetic order in the definition of substitution (Section 4), and for a clear presentation of results like Theorem 7.6.
With rule 7 we want to express that the order of the t_is in $(t_1,\ldots,t_n)$ and the alphabetic order of the variables are not characteristics of the Principia, but are only introduced for the technical reasons explained in this remark. This is worked out in Corollary 7.7.

Remark 3.12 As already announced after Definition 3.4, the rôle of orders is marginal in RTT. We could leave them away from the judgements $\Gamma\vdash f:t^a$, as they are a function of the derivation of $\Gamma\vdash f:t^a$ (see also Lemma 7.8). We do not leave them out as we need them in Section 10.

Remark 3.13 In rule 2, we make the assumption that the variables of Γ all must come before the variables of Δ. We must make this assumption, considering the following example:
We can derive

$$\{\mathtt{x}_1:\mathtt{o}^0\}\vdash \mathtt{R}(\mathtt{x}_1,\mathtt{x}_1):(\mathtt{o}^0)^1 \text{ and } \{\mathtt{x}_1:\mathtt{o}^0,\mathtt{x}_2:\mathtt{o}^0\}\vdash \mathtt{R}(\mathtt{x}_1,\mathtt{x}_2):(\mathtt{o}^0,\mathtt{o}^0)^1.$$

Without making the assumption, we would be allowed to derive

$$\{\mathtt{x}_1:\mathtt{o}^0,\mathtt{x}_2:\mathtt{o}^0\}\vdash \mathtt{R}(\mathtt{x}_1,\mathtt{x}_1)\vee\mathtt{R}(\mathtt{x}_1,\mathtt{x}_2):(\mathtt{o}^0,\mathtt{o}^0,\mathtt{o}^0)^1.$$

But the intended meaning of the last-mentioned judgement will appear to be (cf. the forthcoming theorems 7.6 and 7.14): In the context Γ, the pseudomatrix $\mathtt{R}(\mathtt{x}_1,\mathtt{x}_1)\vee\mathtt{R}(\mathtt{x}_1,\mathtt{x}_2)$ has three(!) free variables, which are all of type $\mathtt{o}^0$. And this is not true, of course. If we want to have a determined way to derive the "correct" (in the sense of theorems 7.6 and 7.14) type of $f\vee g$ directly from the types of f and g, we must demand that Γ and Δ do not overlap. For technical reasons (the order of the t_i^a's; see also Theorem 7.6) we even make the assumption that for $x\in\mathrm{dom}(\Gamma)$ and $y\in\mathrm{dom}(\Delta)$, $x<y$ must hold.

As Russell and Whitehead do not have a formal notation for types, they do not forbid this kind of constructions in [16]. In 8.10 we show that our limitation to non-overlapping contexts as made in rule 2 is not a real limitation: still all the desired judgements can be derived.

Remark 3.14 The notion of "replacing" can be made formal: Replacing g bij z in rule 3 can be read as replacing the tree that represents g by the leave z; replacing g bij $z(x_1, \ldots, x_p)$ can be read as replacing the tree that represents g by the tree that represents $z(x_1, \ldots, x_p)$.

Remark 3.15 Contexts as used in RTT (3.9) contain, in a sense, too much information: not only information on free variables, but also information on non-free variables. Hence, the contexts are not as "elegant" as one would desire. We will discuss this in the second part of Section 9, where we also propose a solution to this problem. For the moment, we only prove the following lemma.

Lemma 3.16 *Let Γ be a context, f a pseudomatrix, and assume $\Gamma \vdash f : t^a$. Then*

1. *All variables of f (free and non-free) are in* $\mathrm{dom}(\Gamma)$*;*
2. *If Δ is the (unique) subset of Γ such that* $\mathrm{dom}(\Delta)$ *contains exactly all the variables of f, then $\Delta \vdash f : t^a$.*

PROOF: An easy induction on the definition of $\Gamma \vdash f : t^a$. □

Part 1 of this lemma shows that the contexts Γ' in 3.9.3, 3.9.4 and 3.9.6 really exist.

Later we will need the idea of *major* and *minor* premises. We define:

Definition 3.17 The premises $\Gamma \vdash f : (t_1^{a_1}, \ldots, t_n^{a_n})^a$ and $\Delta \vdash g : (u_1^{b_1}, \ldots, u_m^{b_m})^b$ of rule 2, the premises $\Gamma \vdash f : (t_1^{a_1}, \ldots, t_m^{a_m})^a$ of rule 3 and 4, the premise $\Gamma \vdash f : t^m$ of rule 5 and the premises $\Gamma \cup \{y : t_i^{a_i}\} \vdash f : (t_1^{a_1}, \ldots, t_n^{a_n})^a$ of rule 6 and rule 7 are called *major* premises.
The premises $\Gamma \vdash g : t_{m+1}^{a_{m+1}}$ of rule 3 and 4 and the premise $\Gamma \vdash k : t_i^{a_i}$ of rule 6 are called *minor* premises.

Example 3.18 *We will give some examples, in order to illustrate how our system works. We will use a notation of the form* $\dfrac{X_1, \ldots, X_n}{Y} N$*, indicating that from the judgements $X_1, \ldots, X_n$, we can infer the judgement Y by using rule N.*

- $$\vdash \mathtt{R}(\mathtt{a}_1, \mathtt{a}_2) : ()^0$$

- $$\frac{\vdash \mathtt{R}_1(\mathtt{a}_1) : ()^0 \qquad \vdash \mathtt{R}_2(\mathtt{a}_1) : ()^0}{\vdash \mathtt{R}_1(\mathtt{a}_1) \vee \mathtt{R}_2(\mathtt{a}_1) : ()^0}\,2$$

  but not*:*

  $$\frac{\mathtt{x}_1 : \mathtt{o}^0 \vdash \mathtt{R}_1(\mathtt{x}_1) : (\mathtt{o}^0)^1 \qquad \mathtt{x}_1 : \mathtt{o}^0 \vdash \mathtt{R}_2(\mathtt{x}_1) : (\mathtt{o}^0)^1}{\mathtt{x}_1 : \mathtt{o}^0 \vdash \mathtt{R}_1(\mathtt{x}_1) \vee \mathtt{R}_2(\mathtt{x}_1) : (\mathtt{o}^0, \mathtt{o}^0)^1}\,2$$

  To obtain $\mathtt{R}_1(\mathtt{x}_1) \vee \mathtt{R}_2(\mathtt{x}_1)$ *we must make a different start:*

  $$\frac{\dfrac{\dfrac{\vdash \mathtt{R}_1(\mathtt{a}_1) : ()^0 \qquad \vdash \mathtt{R}_2(\mathtt{a}_1) : ()^0}{\vdash \mathtt{R}_1(\mathtt{a}_1) \vee \mathtt{R}_2(\mathtt{a}_1) : ()^0}\,2 \qquad \vdash \mathtt{a}_1 : \mathtt{o}^0}{\mathtt{x}_1 : \mathtt{o}^0 \vdash \mathtt{R}_1(\mathtt{x}_1) \vee \mathtt{R}_2(\mathtt{x}_1) : (\mathtt{o}^0)^1}\,3$$

- $$\frac{\begin{array}{ll} \mathtt{x}_1 : \mathtt{o}^0, \mathtt{x}_2 : \mathtt{o}^0 & \vdash \ \mathtt{R}_1(\mathtt{x}_1) \vee \neg \mathtt{R}_2(\mathtt{x}_1, \mathtt{x}_1, \mathtt{x}_2) : (\mathtt{o}^0, \mathtt{o}^0)^1 \\ \mathtt{x}_1 : \mathtt{o}^0, \mathtt{x}_2 : \mathtt{o}^0 & \vdash \ \mathtt{R}_1(\mathtt{x}_1) : (\mathtt{o}^0)^1 \end{array}}{\mathtt{x}_1 : \mathtt{o}^0, \mathtt{x}_2 : \mathtt{o}^0, \mathtt{x}_3 : (\mathtt{o}^0)^1 \vdash \mathtt{x}_3(\mathtt{x}_1) \vee \neg \mathtt{R}_2(\mathtt{x}_1, \mathtt{x}_1, \mathtt{x}_2) : (\mathtt{o}^0, \mathtt{o}^0, (\mathtt{o}^0)^1)^2}\,4$$

- $$\frac{\begin{array}{ll} \mathtt{x}_1 : \mathtt{o}^0, \mathtt{x}_2 : \mathtt{o}^0 & \vdash \ \mathtt{R}_1(\mathtt{x}_1) \vee \neg \mathtt{R}_2(\mathtt{x}_1, \mathtt{x}_1, \mathtt{x}_2) : (\mathtt{o}^0, \mathtt{o}^0)^1 \\ \mathtt{x}_1 : \mathtt{o}^0, \mathtt{x}_2 : \mathtt{o}^0 & \vdash \ \mathtt{R}_2(\mathtt{x}_1, \mathtt{x}_1, \mathtt{x}_2) : (\mathtt{o}^0, \mathtt{o}^0)^1 \end{array}}{\mathtt{x}_1 : \mathtt{o}^0, \mathtt{x}_2 : \mathtt{o}^0, \mathtt{x}_3 : (\mathtt{o}^0, \mathtt{o}^0)^1 \vdash \mathtt{R}_1(\mathtt{x}_1) \vee \neg \mathtt{x}_3(\mathtt{x}_1, \mathtt{x}_2) : (\mathtt{o}^0, \mathtt{o}^0, (\mathtt{o}^0, \mathtt{o}^0)^1)^2}\,4$$

•
$$\frac{\begin{array}{c}x_1 : 0^0, x_2 : 0^0 \ \vdash \ R_1(x_1) \vee \neg R_2(x_1, x_1, x_2) : (0^0, 0^0)^1 \\ x_1 : 0^0, x_2 : 0^0 \ \vdash \ \neg R_2(x_1, x_1, x_2) : (0^0, 0^0)^1\end{array}}{x_1 : 0^0, x_2 : 0^0, x_3 : (0^0, 0^0)^1 \vdash R_1(x_1) \vee x_3(x_1, x_2) : (0^0, 0^0, (0^0, 0^0)^1)^2}\,4$$

•
$$\frac{\begin{array}{c}x_1 : 0^0, x_2 : 0^0 \ \vdash \ R_1(x_1) \vee \neg R_2(x_1, x_1, x_2) : (0^0, 0^0)^1 \\ x_1 : 0^0, x_2 : 0^0 \ \vdash \ R_1(x_1) \vee \neg R_2(x_1, x_1, x_2) : (0^0, 0^0)^1\end{array}}{x_1 : 0^0, x_2 : 0^0, x_3 : (0^0, 0^0)^1 \vdash x_3(x_1, x_2) : (0^0, 0^0, (0^0, 0^0)^1)^2}\,4$$

•
$$\frac{x_1 : 0^0 \vdash R_1(x_1) \vee R_2(x_1) : (0^0)^1 \qquad \vdash a_1 : 0^0}{\vdash R_1(a_1) \vee R_2(a_1) : ()^0}\,6$$

•
$$\frac{\begin{array}{c}x_1 : 0^0, x_2 : 0^0, x_3 : (0^0, 0^0)^1 \ \vdash \ R_1(x_1) \vee \neg x_3(x_1, x_2) : (0^0, 0^0, (0^0, 0^0)^1)^2 \\ x_1 : 0^0, x_2 : 0^0 \ \vdash \ R_2(x_1, x_1, x_2) : (0^0, 0^0)^1\end{array}}{x_1 : 0^0, x_2 : 0^0 \vdash R_1(x_1) \vee \neg R_2(x_1, x_1, x_2) : (0^0, 0^0)^1}\,6$$

Example 3.19 *Rule 6 is problematic. The following example shows that the idea of substitution used in the Principia is difficult. Consider the derivation:*

$$\frac{\begin{array}{c}x_1 : 0^0, x_2 : (0^0, (0^0)^1)^2, x_3 : 0^0, x_4 : (0^0)^1 \ \vdash \ x_2(x_1, R(x_1)) : (0^0, (0^0, (0^0)^1)^2)^3 \\ x_1 : 0^0, x_3 : 0^0, x_4 : (0^0)^1 \ \vdash \ x_4(x_3) : (0^0, (0^0)^1)^2\end{array}}{x_1 : 0^0 \vdash R(x_1) : (0^0)^1}\,6$$

It is not very clear what happens here. Some substitution must have taken place, but we can not trace how it has taken place, as we do not have a good notion of substitution. However, this example turns out to be all right (see Example 4.9) after we have formalized substitution.

Remark 3.20 We would like to prove some theorems now, for instance that if (for a certain context Γ and a pseudomatrix f) $\Gamma \vdash f : (t_1^{a_1}, \ldots, t_n^{a_n})^a$, and $x_1 < x_2 < \ldots < x_m$ are the free variables of f, then $m = n$ and $x_i : t_i^{a_i} \in \Gamma$ for all i.
The proof should be given by induction, and indeed, the cases 1–5 are easy to prove. But to prove the cases 6 and 7 we are obliged, again, to take a closer look on what substitution actually *is*. This will be done in the next sections.

4 Substitution

As promised, we will take a closer look on substitution in this section. We would prefer to take a definition of substitution that is given in the Principia, if there was such a definition. Unfortunately, such a definition is not presented. We therefore have to rely on our intuition, and on the intuition behind pseudomatrices as was discussed in remarks 2.2 and 2.4, which relies on our interpretation of what is meant in the Principia.

Definition 4.1 Let $f \in \mathcal{P}$ and $g_1, \ldots, g_n \in \mathcal{A} \cup \mathcal{V} \cup \mathcal{P}$.
Suppose $x_1, \ldots, x_n \in \mathcal{V}$ are distinct variables. We define the notion of a *substitution-to-carry-out* (or shortly *substitution* if no confusion arises) by induction on f:

1. If $f = R(i_1, \ldots, i_{a(R)})$ then $f\{x_1{:=}g_1, \ldots, x_n{:=}g_n\}$ is a substitution;
2. Assume $f = z(k_1, \ldots, k_m)$, and assume also that if ($z = x_i$ and $k_q = x_j$), then $g_i \neq g_j$. Then $f\{x_1{:=}g_1, \ldots, x_n{:=}g_n\}$ is a substitution;

3. Assume, $f = f_1 \vee f_2$ and $f_k\{x_1{:=}g_1, \ldots, x_n{:=}g_n\}$ is a substitution ($k = 1, 2$). Then $f\{x_1{:=}g_1, \ldots, x_n{:=}g_n\}$ is a substitution.
   Assume, $f = \neg f'$ and $f'\{x_1{:=}g_1, \ldots, x_n{:=}g_n\}$ is a substitution.
   Then $f\{x_1{:=}g_1, \ldots, x_n{:=}g_n\}$ is a substitution.

We will shortly write $f\{x_i{:=}g_i\}$ for $f\{x_1{:=}g_1, \ldots, x_n{:=}g_n\}$, as long as no confusion arises.

Remark 4.2 The condition in point 2, that ($z = x_i$ and $k_q = x_j$) must imply $g_i \neq g_j$ is a reasonable condition if we look at the Vicious Circle Principle 1.1. If we had $z = x_i$, $k_q = x_j$ and $g_i = g_j$ then we would allow to substitute exactly the same for z as for k_q, and this would result in a propositional function that is substituted in itself. This kind of constructions is forbidden by the vicious circle principle, so it is not a real limitation to forbid them in our system, too.
We will need this limitation to prove an important technical result on substitution (Theorem 6.3). That the limitation is really needed is showed in Example 6.4.

Remark 4.3 Note that pseudomatrices are also substitutions (take $n = 0$). We will call them *empty* substitutions.

We will call some substitutions easy, and other substitutions hard:

Definition 4.4 Let $S = f\{x_i{:=}g_i\}$ be a substitution.

- If $f = R(i_1, \ldots, i_{a(R)})$ then S is called easy;
- Assume $f = z(k_1, \ldots, k_m)$. If, for some i, $z = x_i$ and $g_i \in \mathcal{P}$ then S is called hard; otherwise S is called easy;
- Assume $f = f_1 \vee f_2$. S is called easy if both $f_1\{x_i{:=}g_i\}$ and $f_2\{x_i{:=}g_i\}$ are easy; otherwise S is called hard.
  Assume $f = \neg f'$. S is called easy if $f'\{x_i{:=}g_i\}$ is easy; otherwise S is called hard.

Our goal is now to describe in which way substitutions can be carried out. In order to reach this goal we need the notion of a *substitution tree*.

Definition 4.5 (Substitution-trees)

1. Each substitution is a substitution-tree.
2. If T_1 and T_2 are substitution-trees, then also

   are substitution trees. We simply write $T_1 \vee T_2$ and $\neg T_1$ for these trees.
3. All substitution trees can be obtained by using the construction rules 1 and 2.

A substitution tree T is in *normal form* if T consists only of empty substitutions (and the symbols $\vee, \neg$).

Note that such a substitution tree represents a pseudomatrix, and that a substitution tree which is *not* in normal form does *not* represent a pseudomatrix.

We now define a notion of *reduction* on substitution trees. Reducing a substitution tree S to another substitution tree must be seen as making one step in carrying out the substitution S.

Definition 4.6 Let S be a substitution tree. We define the notion $S \longrightarrow T$ in an inductive way:

1. Suppose S is a substitution, say: $S = f\{x_1{:=}g_1, \ldots, x_n{:=}g_n\}$.

(a) Assume $f = R(i_1, \ldots, i_{a(R)})$ for some $R \in \mathcal{R}$ and $i_1, \ldots, i_{a(R)} \in \mathcal{A} \cup \mathcal{V}$.
Observe the list of symbols $R(i'_1, \ldots, i'_{a(R)})$, where

- $i'_j = i_j$ if $i_j \notin \{x_1, \ldots, x_n\}$;
- $i'_j = g_k$ if $i_j = x_k$.

If this list of symbols is a pseudomatrix[21], then we say that S reduces to $R(i'_1, \ldots, i'_{a(R)})$ and write $S \longrightarrow R(i'_1, \ldots, i'_{a(R)})$.

(b) Assume $f = z(k_1, \ldots, k_m)$ for a $z \in \mathcal{V}$ and $k_1, \ldots, k_m \in \mathcal{A} \cup \mathcal{V} \cup \mathcal{P}$.
If S is *easy* then observe the list of symbols $z'(k'_1, \ldots, k'_m)$, where

- $k'_j = k_j$ if $k_j \notin \{x_1, \ldots, x_n\}$;
- $k'_j = g_k$ if $k_j = x_k$;
- $z' = z$ if $z \notin \{x_1, \ldots, x_n\}$;
- $z' = g_k$ if $z = x_k$.

If this list of symbols is a pseudomatrix[22] then we say that S reduces to $z'(k'_1, \ldots, k'_m)$ and write $S \longrightarrow z'(k'_1, \ldots, k'_m)$.
If S is *hard*, then first determine $z'(k'_1, \ldots, k'_m)$, as in the case of the "easy" substitution. Note that $z' \in \mathcal{P}$ because the substitution is hard. We proceed in the line of the remarks 2.2 and 2.4:
If $\#\text{FV}(z') = m$, and $y_1 < \ldots < y_m$ are the free variables of z', and $z'\{y_i{:=}k'_i\}$ is a substitution, then we say that S reduces to $z'\{y_i{:=}k'_i\}$.
We write $S \longrightarrow z'\{y_i{:=}k'_i\}$, or, if we want to emphasize the two steps of the proces described above, $S \rightsquigarrow_1 z'(k'_1, \ldots, k'_m) \rightsquigarrow_2 z'\{y_i{:=}k'_i\}$.

(c) If $f = f_1 \vee f_2$, and $f_1\{x_i{:=}g_i\} \longrightarrow T_1$, and $f_2\{x_i{:=}g_i\} \longrightarrow T_2$, then $S \longrightarrow T_1 \vee T_2$;
if $f = \neg f'$ and $f'\{x_i{:=}g_i\} \longrightarrow T$, then $S \longrightarrow \neg T$.

2. Now assume $S = T_1 \vee T_2$, where T_1 and T_2 are substitution trees.
If U_1 is a substitution tree and $T_1 \longrightarrow U_1$ then $S \longrightarrow U_1 \vee T_2$.
If U_2 is a substitution tree and $T_2 \longrightarrow U_2$, then $S \longrightarrow T_1 \vee U_2$.
If $S = \neg T$, where T is a substitution tree, and $T \longrightarrow U$, then $S \longrightarrow \neg U$.

Remark 4.7 It is possible that a substitution tree that is not in normal form, doesn't reduce to any other substitution tree any more. Take, for example, the substitution $\mathtt{x(y)\{x{:=}R(p,q,r)\}}$. This is a hard substitution, and $\mathtt{x(y)\{x{:=}R(p,q,r)\}} \rightsquigarrow_1 \mathtt{R(p,q,r)(y)}$. As the number of free variables in $\mathtt{R(p,q,r)}$ is not equal to the number of arguments of $\mathtt{x(y)}$, we cannot carry out the reduction as described in 4.6.1(b).

Remark 4.8 Note that a substitution tree S can reduce to at most two different substitution trees T_1 and T_2. Moreover, if $S \longrightarrow T_1$ and $S \longrightarrow T_2$ then S must be of the form $S_1 \vee S_2$ and the reduction of S to T_1 (T_2) can change only one of the S_i's, so that there is a substitution tree U such that $T_1 \longrightarrow U$ and $T_2 \longrightarrow U$.
With this in mind it is easy to prove a Church-Rosser-like theorem for the reduction $\longrightarrow$.

Example 4.9 *We give some concrete examples to make the notion $S \longrightarrow T$ more clear.*

1. $\mathtt{R(x_1,x_3)\{x_1{:=}a_1\}} \longrightarrow \mathtt{R(a_1,x_3)}$;
2. $\mathtt{R(x_1,x_3)\{x_2{:=}a_2\}} \longrightarrow \mathtt{R(x_1,x_3)}$;
3. $\mathtt{z_1(R(x_1,x_2),x_3,a_2)\{x_1{:=}x_4\}} \longrightarrow \mathtt{z_1(R(x_1,x_2),x_3,a_2)}$ *(note that* $\mathtt{x_1}$ *is* not *a free variable!)*;

[21] Note that this happens exactly if there is no j such that $x_j \in \text{FV}(f)$ and $g_j \in \mathcal{P}$.
[22] And this happens exactly if $z \notin \{x_1, \ldots, x_n\}$, or $z = x_i$ and $g_i \notin \mathcal{A}$. Note that $g_i \notin \mathcal{P}$, as the substitution was easy.

4. $\mathtt{z_1(x_3,a_2,R(x_1,x_2))\{z_1{:=}z_2(x_4,x_1),x_3{:=}a_1\}}$
$\leadsto_1 \mathtt{(z_2(x_4,x_1))(a_1,a_2,R(x_1,x_2))}$
$\leadsto_2 \mathtt{z_2(x_4,x_1)\{x_1{:=}a_1,x_4{:=}a_2,z_2{:=}R(x_1,x_2)\}}$
$\leadsto_1 \mathtt{(R(x_1,x_2))(a_2,a_1)}$
$\leadsto_2 \mathtt{R(x_1,x_2)\{x_1{:=}a_2,x_2{:=}a_1\}}$
$\longrightarrow \mathtt{R(a_2,a_1)}$;

5. $\mathtt{(z_1(x_3,a_2,R(x_1,x_2)) \vee z_2(x_3))\{z_1{:=}z_2(x_4,x_1),z_2{:=}R(x_1,x_1)\}}$
$\leadsto_1 \mathtt{(z_2(x_4,x_1))(x_3,a_2,R(x_1,x_2)) \vee (R(x_1,x_1))(x_3)}$
$\leadsto_2 \mathtt{z_2(x_4,x_1)\{x_1{:=}x_3,x_4{:=}a_2,z_2{:=}R(x_1,x_2)\} \vee R(x_1,x_1)\{x_1{:=}x_3\}}$
$(*) \leadsto_1 \mathtt{(R(x_1,x_2))(a_2,x_3) \vee R(x_1,x_1)\{x_1{:=}x_3\}}$
$(*) \leadsto_2 \mathtt{R(x_1,x_2)\{x_1{:=}a_2,x_2{:=}x_3\} \vee R(x_1,x_1)\{x_1{:=}x_3\}}$
$(*) \longrightarrow \mathtt{R(a_2,x_3) \vee R(x_1,x_1)\{x_1{:=}x_3\}}$
$(**) \longrightarrow \mathtt{R(a_2,x_3) \vee R(x_3,x_3)}$.

At $(*)$ we reduce the left part of the expression; at $(**)$ we reduce the right part.

6. *We work out the substitution of Example 3.18:*
$\mathtt{x_2(x_1,R(x_1))\{x_2{:=}x_4(x_3)\}}$
$\leadsto_1 \mathtt{(x_4(x_3))(x_1,R(x_1))}$
$\leadsto_2 \mathtt{x_4(x_3)\{x_3{:=}x_1,x_4{:=}R(x_1)\}}$
$\leadsto_1 \mathtt{(R(x_1))(x_1)}$
$\leadsto_2 \mathtt{R(x_1)\{x_1{:=}x_1\}}$
$\longrightarrow \mathtt{R(x_1)}$

5 Intermezzo: substitution in λ-notation

Our notion of reduction has something to do with the idea of β-reduction in λ-calculus. If we want to illustrate this, we need a different way of writing down our pseudomatrices:

Definition 5.1

- We will write $Ri_1 \ldots i_{a(R)}$ instead of $R(i_1, \ldots, i_{a(R)})$.
- If $z \in \mathcal{V}$ and $k_1, \ldots, k_m \in \mathcal{A} \cup \mathcal{V} \cup \mathcal{P}$ then we write $zk'_1 \ldots k'_m$ instead of $z(k_1, \ldots, k_m)$. Here, $k'_i = k_i$ if $k_i \in \mathcal{A} \cup \mathcal{V}$.
If $k_i \in \mathcal{P}$, then first determine the "new notation" for k_i, say, ℓ_i. Further, assume that $x_1 < \ldots < x_j$ are the free variables of k_i. Now define $k'_i = \lambda x_1 \ldots x_j.\ell_i$.
- If $f = f_1 \vee f_2$, then first determine the "new notation" for f_1 and f_2, say: g_1 and g_2. The new notation for f will be: $\vee g_1 g_2$.
If $f = \neg f'$, then first determine the "new notation" for f', say: g. The new notation for f will be: $\neg g$.

If f is a pseudomatrix (in the old notation), we will write $\tilde{f}$ for this pseudomatrix in new notation.

Example 5.2

f	$\tilde{f}$
$\mathtt{R(x_1,x_3)}$	$\mathtt{Rx_1x_3}$
$\mathtt{z_1(R(x_1,x_2),x_3,a_2)}$	$\mathtt{z_1(\lambda x_1x_2.Rx_1x_2)x_3a_2}$
$\mathtt{z_1(x_3,a_2,R(x_1,x_2)) \vee z_2(x_3)}$	$\mathtt{\vee(z_1x_3a_2(\lambda x_1x_2.Rx_1x_2))(z_2x_3)}$
$\mathtt{z_1(x_3,a_2,z_2(R(x_1,x_1)))}$	$\mathtt{z_1x_3a_2(\lambda z_2.z_2(\lambda x_1.Rx_1x_1))}$

Remark 5.3 The notion of free variable (cf. Remark 2.4) of a pseudomatrix f is exactly the notion of free variable in the lambda term $\tilde{f}$.

Remark 5.4 The adding of the abstraction-λs in Definition 5.1 could be paraphrased as follows: the intuition of Whitehead and Russell apparently was that all pseudomatrices f occurring as an *argument* in another pseudomatrix, should be considered to have been *closed* with respect to their free variables.

Definition 5.5 A substitution $f\{x_i{:=}g_i\}$ can be written as $(\lambda x_1 \ldots x_n.\tilde{f})\widetilde{g_1}' \ldots \widetilde{g_n}'$.
Here, $\widetilde{g_i}' = g_i$ if $g_i \in \mathcal{A} \cup \mathcal{V}$; otherwise $\widetilde{g_i}' = \lambda y_1 \ldots y_m.\tilde{g_i}$, where $y_1 < \ldots < y_m$ are the free variables of g_i.

There is a small difference between our notion of reduction and β-reduction. β-reduction reduces $(\lambda x_1 \ldots x_n.\tilde{f})\widetilde{g_1}' \ldots \widetilde{g_n}'$ step by step:

$$\begin{array}{ll} & (\lambda x_1 \ldots x_n.\tilde{f})\widetilde{g_1}' \ldots \widetilde{g_n}' \\ \longrightarrow_\beta & (\lambda x_2 \ldots x_n.\tilde{f}[x_1{:=}\widetilde{g_1}'])\widetilde{g_2}' \ldots \widetilde{g_n}' \\ \longrightarrow_\beta & \ldots \\ \longrightarrow_\beta & (\ldots(\tilde{f}[x_1{:=}\widetilde{g_1}'])[x_2{:=}\widetilde{g_2}'] \ldots)[x_n{:=}\widetilde{g_n}'], \end{array}$$

while our reduction reduces all the occurring redexes *at the same time*, and moreover: all the redexes *have* to be reduced at the same time: It is impossible to reduce the redex $\lambda x_1(\ldots)\widetilde{g_1}' \ldots$ while not reducing the other redexes. If we append β-reduction in this way, it is in line with the reduction of our system, as can be seen in the following examples, and will be proved in Section 9. These examples are exact the same examples as in 4.9, but now written in the new notation with λ's. Observe that the notation with $\rightsquigarrow_1$'s and $\rightsquigarrow_2$'s is no longer necessary!

Example 5.6

1. $(\lambda x_1.\mathtt{R}x_1x_3)\mathtt{a}_1 \longrightarrow \mathtt{R}\mathtt{a}_1x_3$;
2. $(\lambda x_2.\mathtt{R}x_1x_3)\mathtt{a}_1 \longrightarrow \mathtt{R}x_1x_3$;
3. $(\lambda x_1.z_1(\lambda x_1x_2.\mathtt{R}x_1x_2)x_3\mathtt{a}_2)x_4 \longrightarrow z_1(\lambda x_1x_2.\mathtt{R}x_1x_2)x_3\mathtt{a}_2$;
4. $(\lambda z_1x_3.z_1x_3\mathtt{a}_2(\lambda x_1x_2.\mathtt{R}x_1x_2))(\lambda x_1x_4z_2.z_2x_4x_1)\mathtt{a}_1 \longrightarrow$
   $(\lambda x_1x_4z_2.z_2x_4x_1)\mathtt{a}_1\mathtt{a}_2(\lambda x_1x_2.\mathtt{R}x_1x_2) \longrightarrow$
   $(\lambda x_1x_2.\mathtt{R}x_1x_2)\mathtt{a}_2\mathtt{a}_1 \longrightarrow$
   $\mathtt{R}\mathtt{a}_2\mathtt{a}_1$.
5. $(\lambda z_1z_2.\vee(z_1x_3\mathtt{a}_2(\lambda x_1x_2.\mathtt{R}x_1x_2))(z_2x_3))(\lambda x_1x_4z_2.z_2x_4x_1)(\lambda x_1.\mathtt{R}x_1x_1) \longrightarrow$
   $\vee((\lambda x_1x_4z_2.z_2x_4x_1)x_3\mathtt{a}_2(\lambda x_1x_2.\mathtt{R}x_1x_2))((\lambda x_1.\mathtt{R}x_1x_1)x_3) \longrightarrow$
   $\vee((\lambda x_1x_2.\mathtt{R}x_1x_2)\mathtt{a}_2x_3)((\lambda x_1.\mathtt{R}x_1x_1)x_3) \longrightarrow$
   $\vee(\mathtt{R}\mathtt{a}_2x_3)((\lambda x_1.\mathtt{R}x_1x_1)x_3) \longrightarrow$
   $\vee(\mathtt{R}\mathtt{a}_2x_3)(\mathtt{R}x_3x_3)$.
6. $(\lambda x_2.x_2x_1(\lambda x_1.\mathtt{R}x_1))(\lambda x_3x_4.x_4x_3) \longrightarrow$
   $(\lambda x_3x_4.x_4x_3)x_1(\lambda x_1.\mathtt{R}x_1) \longrightarrow$
   $(\lambda x_1.\mathtt{R}x_1)x_1 \longrightarrow$
   $\mathtt{R}x_1$

We will discuss this and other advantages of the λ-notation more extensively in Section 9.

6 The finiteness of reduction paths

Definition 6.1 (Reduction path) A *reduction path* is a sequence of substitution trees $S_0, S_1, \ldots$ (finite or infinite) such that for all $n \in \mathbb{N}$, $S_n \longrightarrow S_{n+1}$, and S_n is not in normal form. If $S_0, \ldots, S_n$ is a finite reduction path, then n is called the *length* of this reduction path.

We now want to prove that there are no infinite reduction paths, i.e. that if we start to carry out a substitution, then our job will end either with the result of the substitution, or with the message that the substitution cannot be carried out (see Remark 4.7). To prove this, we need

Definition 6.2 (Depth) We define the *depth* $\mathfrak{D}(f)$ of a pseudomatrix f by induction. We make the convention that $\sum_{x \in \emptyset} \varphi(x) \stackrel{\text{def}}{=} 0$.

- $\mathfrak{D}(R(i_1, \ldots, i_{a(R)})) \stackrel{\text{def}}{=} 0$;
- $\mathfrak{D}(z(k_1, \ldots, k_n)) \stackrel{\text{def}}{=} 1 + \sum_{k \in \{k_i : k_i \in \mathcal{P}\}} \mathfrak{D}(k)$;
- $\mathfrak{D}(f \vee g) \stackrel{\text{def}}{=} \mathfrak{D}(f) + \mathfrak{D}(g)$;
  $\mathfrak{D}(\neg f) \stackrel{\text{def}}{=} \mathfrak{D}(f)$.

Note, that

$$\mathfrak{D}(z_1(z_2(x_1), z_2(x_1))) = 1 + \mathfrak{D}(z_2(x_1))$$

and *not* $\mathfrak{D}(z_1(z_2(x_1), z_2(x_1))) = 1 + 2 \cdot \mathfrak{D}(z_2(x_1))$!
We can extend this definition to substitutions:

$$\mathfrak{D}(f\{x_i := g_i\}) \stackrel{\text{def}}{=} \mathfrak{D}(f) + \sum_{g \in \{g_i : g_i \in \mathcal{P}\}} \mathfrak{D}(g).$$

Note that (regarding empty substitutions) this definition agrees with the previous definition. Furthermore, we can extend the definition to substitution trees:

$$\mathfrak{D}(T_1 \vee T_2) \stackrel{\text{def}}{=} \mathfrak{D}(T_1) + \mathfrak{D}(T_2);$$

$$\mathfrak{D}(\neg T) \stackrel{\text{def}}{=} \mathfrak{D}(T).$$

If T is a substitution tree in normal form, then this definition agrees with the two previous ones.

Theorem 6.3 *There are no infinite reduction paths.*

PROOF: Assume $S = S_0, S_1, \ldots$ is a reduction path. We will use induction on $\mathfrak{D}(S)$.

- If $\mathfrak{D}(S) = 0$ then the occurring substitutions must be easy. Therefore, S_n must be in normal form for some n, so that the reduction path ends with S_n and is finite.
- Assume that all reduction paths starting with a substitution tree S' such that $\mathfrak{D}(S') \leq n$ are finite (**IH**), and assume that $\mathfrak{D}(S) = n + 1$. Use induction on the substitution tree S:
  1. S is a substitution $f\{x_1 := g_1, \ldots, x_n := g_n\}$.
     (a) $f = R(i_1, \ldots, i_{a(R)})$. This substitution is easy, so the reduction path is finite.
     (b) $f = z(k_1, \ldots, k_m)$. We can assume that the substitution is hard, say:

$$z(k_1, \ldots, k_m)\{x_i := g_i\} \rightsquigarrow_1 z'(k'_1, \ldots, k'_m)$$

Assume $y_1 < \ldots < y_m$ are the free variables of z' in alphabetic order.
If $z'\{y_i := k'_i\}$ is not a substitution then the reduction path ends at S_0; otherwise:

$$\begin{aligned}
\mathfrak{D}(z'\{y_i := k'_i\}) &= \mathfrak{D}(z') + \sum_{k \in \{k'_i : k'_i \in \mathcal{P}\}} \mathfrak{D}(k) \\
&\overset{(*)}{\leq} \mathfrak{D}(z') + \sum_{\substack{g \in \{g_i : g_i \in \mathcal{P}\} \\ g \not\equiv x'}} \mathfrak{D}(g) + \sum_{k \in \{k_i : k_i \in \mathcal{P}\}} \mathfrak{D}(k) \\
&\overset{(**)}{=} \sum_{g \in \{g_i : g_i \in \mathcal{P}\}} \mathfrak{D}(g) + \sum_{k \in \{k_i : k_i \in \mathcal{P}\}} \mathfrak{D}(k) \\
&< 1 + \sum_{g \in \{g_i : g_i \in \mathcal{P}\}} \mathfrak{D}(g) + \sum_{k \in \{k_i : k_i \in \mathcal{P}\}} \mathfrak{D}(k) \\
&= \mathfrak{D}(z(k_1, \ldots, k_m)\{x_i := g_i\})
\end{aligned}$$

At (*), we use that $z' \in \{k_i' : k_i' \in \mathcal{P}\} \Rightarrow z' \in \{k_i : k_i \in \mathcal{P}\}$. This is because of the condition that $z = x_i$ and $k_q = x_j$ implies $g_i \neq g_j$. Therefore, $\{k_i' : k_i' \in \mathcal{P}\} = \{g_i : g_i \neq z', g_i \in \mathcal{P}\} \cup \{k_i : k_i \in \mathcal{P}\}$.
The equality at (**) holds because the substitution is hard, so $z' = g_i$ holds for some $g_i \in \mathcal{P}$.
So, by (**IH**), a reduction path starting with $S_1 = z'\{y_i{:=}k_i'\}$ is finite, and therefore the path starting with S is finite, too.

(c) Assume f_1 and f_2 are pseudomatrices, and it has already been proved that

$$\mathfrak{D}(f_q\{y_i{:=}h_i\}) \leq n+1$$

implies that, for $q = 1, 2$, the reduction path of $f_q\{y_i{:=}h_i\}$ is finite.
Now assume $f = f_1 \vee f_2$ and $\mathfrak{D}(f\{x_i{:=}g_i\}) \leq n+1$.
Then $\mathfrak{D}(f_q\{x_i{:=}g_i\}) \leq n+1$ for $q = 1, 2$, so we can conclude that the reduction paths of the $f_q\{x_i{:=}g_i\}$ are finite. But then the reduction path of S is finite, too.
A similar argument holds if $f = \neg f'$.

2. Assume, T_1 and T_2 are substitution trees, and $\mathfrak{D}(T_q) \leq n+1$ implies that the reduction path of T_q is finite, for $q = 1, 2$.
Assume $S = T_1 \vee T_2$. By an argument similar to the argument in 1(c), one shows that the reduction path of S is finite. The same if $S = \neg T$.

- By induction, it is proved that there are no infinite reduction paths.

□

Example 6.4 *As promised in 4.2, we show that the condition $z = x_i$ and $k_q = x_j$ implies $g_i \neq g_j$ in the definition of substitution is* necessary *to obtain the result of the foregoing theorem.*
If we had left out this condition, then

$$S \stackrel{\text{def}}{=} \mathbf{z(x,x)\{x{:=}z(x,x), z{:=}z(x,x)\}}$$

would have been a substitution, and $S, S, S, \ldots$ would have been a reduction path, as $S \longrightarrow S$, and S is not in normal form. We can compare S in this example with the β-reduction of the lambda term $(\lambda\mathbf{zx.zxx})(\lambda\mathbf{zx.zxx})(\lambda\mathbf{zx.zxx})$ to itself. See also the comparison between substitutions and β-reduction in Section 5 and in lemmas 9.6 and 9.7.

Definition 6.5 If $f\{x_1{:=}g_1, \ldots, x_n{:=}g_n\}$ is a substitution and h is a pseudomatrix, and there is a reduction path of $f\{x_i{:=}g_i\}$ that ends with h, then we will write $f[x_1{:=}g_1, \ldots, x_n{:=}g_n]$, or, shorthand, $f[x_i{:=}g_i]$, for h. We can use this notation because this h, if it exists, is unique (4.8).

Notice that we have not proved that for each substitution $f\{x_i{:=}g_i\}$, $f[x_i{:=}g_i]$ actually *exists*! It is possible that a reduction path does not end in a pseudomatrix.
When we speak about $f[x_i{:=}g_i]$ in the following, we will implicitly assume that there exists a pseudomatrix h such that $h = f[x_i{:=}g_i]$.
In particular, this convention applies to the definition of substitution in RTT (see Definition 3.9.6).

The following lemma is easy to prove and will be useful in the next sections:

Lemma 6.6

- $(f_1 \vee f_2)[y_i{:=}g_i] = (f_1[y_i{:=}g_i]) \vee (f_2[y_i{:=}g_i])$;
- $(\neg f)[y_i{:=}g_i]) = \neg(f[y_i{:=}g_i])$.

Theorem 6.7 (First Free Variable Theorem) *Suppose, $S = f\{x_1{:=}g_1, \ldots, x_n{:=}g_n\}$ is a substitution, and $h = f[x_i{:=}g_i]$.*
Then $\text{FV}(h) = \{g_i \in \mathcal{V} | x_i \in \text{FV}(f)\} \cup (\text{FV}(f) \setminus \{x_1, \ldots, x_n\})$.

PROOF: We will use induction on the length of the reduction path.

- If the length of the reduction path is 0, then $S = f = h$, $n = 0$ and the result is immediate.
- Assume the theorem holds for all reduction paths with length $< N$, and the reduction path from S to h has length N.
  Distinguish two cases:

  f **is simple.** If the substitution is easy, then the result is proved by a simple observation. So assume, the substitution is hard; $f = z(k_1, \ldots, k_m)$, and

  $$S \rightsquigarrow_1 z'(k'_1, \ldots, k'_m) \rightsquigarrow_2 z'\{y_i{:=}k'_i\},$$

  where $y_1 < \ldots < y_m$ are the free variables of z'.
  Note: $z'\{y_i{:=}k'_i\} \longrightarrow h$ has a reduction path of length $N-1$, so we can apply the induction hypothesis on $z'\{y_i{:=}k'_i\}$. Together with the definition of z' and k'_i as explained in 4.6, this enables us to compute $\mathrm{FV}(h)$:

  $$\begin{array}{rcl} \mathrm{FV}(h) & \overset{\mathrm{IH}}{=} & \{k'_i \in \mathcal{V} | y_i \in \mathrm{FV}(z')\} \cup (\mathrm{FV}(z') \setminus \{y_1, \ldots, y_m\}) \\ & \overset{(*)}{=} & \{k'_i \in \mathcal{V} | y_i \in \mathrm{FV}(z')\} \\ & \overset{(*)}{=} & \{k'_i | k'_i \in \mathcal{V}\} \\ & \overset{(**)}{=} & \{g_i \in \mathcal{V} | x_i \in \mathrm{FV}(f)\} \cup \{y \in \mathrm{FV}(f) | y \notin \{x_1, \ldots, x_n\}\} \\ & = & \{g_i \in \mathcal{V} | x_i \in \mathrm{FV}(f)\} \cup (\mathrm{FV}(f) \setminus \{x_1, \ldots, x_n\}) \end{array}$$

  At $(*)$ we use the fact that $\mathrm{FV}(z') = \{y_1, \ldots, y_m\}$.
  At $(**)$ we use the definition of k'_i as given in 4.6.1(b).

  f **is complex.** Say: f consists of the simple pseudomatrices $f_1, \ldots, f_q$.
  Using the result above, and the fact that the reduction paths of the f_i must have length $\leq N$ because the reduction path of f has length N, we find:

  $$\begin{array}{rcl} \mathrm{FV}(h) & = & \bigcup_{j=1}^{q} \{g_i \in \mathcal{V} | x_i \in \mathrm{FV}(f_j)\} \cup (\mathrm{FV}(f_j) \setminus \{x_1, \ldots, x_n\}) \\ & = & \{g_i \in \mathcal{V} | x_i \in \mathrm{FV}(f)\} \cup (\mathrm{FV}(f) \setminus \{x_1, \ldots, x_n\}) \end{array}$$

- The theorem has now been proved for all reduction paths.

□

7 Substitution and types

The notion of substitution has been defined in the previous three sections, and with this definition, the definition of substitution in RTT is complete. There is still one small gap to fill. In the definition of substitution in RTT, it might happen that (notation of Definition 3.9.6) there is no pseudomatrix h such that $h = f[y{:=}k]$. See also the remark after Definition 6.5. The final goal of this section is to prove that such a h always exists. (Corollary 7.18).
In the meantime, some other important characteristics of RTT will be proved (Theorem 7.6, Corollary 7.9 and Theorem 7.17).

We defined substitutions without regarding types. But, for a variable of, say, type t^a (in a certain context Γ) we only want to substitute things that are of the same type (in that context). Therefore, we restrict ourselves to well-typed substitutions:

Definition 7.1 (Well-typed substitution) Let Γ be a context, $S = f\{x_1{:=}g_1, \ldots, x_n{:=}g_n\}$ a substitution.
We call S *well-typed* in the context Γ if $\Gamma \vdash f : t_1^{a_1}$, and for all $i \in \{1, \ldots, n\}$:

- If $g_i \in \mathcal{A}$ then $x_i : \mathsf{0}^{\mathsf{0}} \in \Gamma$;

- If $g_i \in \mathcal{V}$ then there is t^a such that $x_i : t^a \in \Gamma$ and $g_i : t^a \in \Gamma$;
- If $g_i \in \mathcal{P}$ then there is t^a such that $x_i : t^a \in \Gamma$ and $\Gamma \vdash g_i : t^a$.

If we carry out a substitution, substitution trees may occur at some points. Therefore, we want a sensible definition of "well-typed substitution tree", too. Such a tree consists of zero or more non-empty and zero or more empty substitutions. We want, of course, that the non-empty ones are well-typed, but it is not reasonable to demand that the empty ones are, too. These non-empty ones are (part of) the final result of the substitution path, and such a substitution may be carried out during an application of rule 6, just in order to *prove* that this final result is well-typed. Such a proof is completed only after the full substitution has been carried out, therefore the well-typedness of empty substitutions in a substitution tree cannot be demanded.
We come to the following definition:

Definition 7.2 (Well-typed substitution trees) Let Γ be a context.

- Each well-typed substitution in the context Γ is a well-typed substitution tree in the context Γ.
- If T_1 is either a pseudomatrix or a well-typed substitution tree (in Γ), and T_2 is also either a pseudomatrix or a well-typed substitution tree (in Γ), then $T_1 \vee T_2$ is a well-typed substitution tree (in Γ).
- If T is a well-typed substitution tree (in Γ), then $\neg T$ is also a well-typed substitution tree (in Γ).

Definition 7.3 (Well-typed reduction) Let Γ be a context, and S, T substitution trees.
We say that S reduces to T *in a well-typed way* (according to Γ), notation $S \longrightarrow_\Gamma T$, if $S \longrightarrow T$, and S is a well-typed substitution tree in the context Γ.

Definition 7.4 Let Γ be a context. A *well-typed reduction path* in the context Γ is a reduction path $S_0, S_1, \ldots, S_n$ of substitution trees such that for all $m < n$, $S_m \longrightarrow_\Gamma S_{m+1}$.

Definition 7.5 If $S = f\{x_i{:=}g_i\}$ is a well-typed substitution in a context Γ, and there is a well-typed reduction path (according to Γ) from S to a pseudomatrix h, then we write $h = f[x_i{:=}g_i]_\Gamma$.

We must adapt our system of Section 3 a little bit. In rule 6, $f[y{:=}k]$ must be replaced by $f[y{:=}k]_{\Gamma'}$, where $\Gamma' = \Gamma \cup \{y : t_i^{a_i}\}$. In rule 7, $f[y{:=}z]$ must be replaced by $f[y{:=}z]_{\Gamma''}$, where $\Gamma'' = \Gamma \cup \{y{:}t_i^{a_i}, z{:}t_i^{a_i}\}$.
Remark that rule 6 can only be used if there exists a pseudomatrix h such that $h = f[y{:=}k]_{\Gamma'}$. In Theorem 7.17 and Corollary 7.18 we prove that such a h always exists.
The substitutions that occur in rule 7 are always well-typed and can always be carried out.

As we have a good notion of substitution after our work in Section 4 and Section 6, we can prove the theorems that we wanted to prove at the end of Section 3:

Theorem 7.6 (Second Free Variable Theorem) *Assume Γ is a context, f is a pseudomatrix such that $\Gamma \vdash f : (t_1^{a_1}, \ldots, t_n^{a_n})^a$, and $x_1 < \ldots < x_m$ are the free variables of f.*
Then $m = n$ and $x_i : t_i^{a_i} \in \Gamma$ for all $i \leq n$.

PROOF: An easy induction on $\Gamma \vdash f : (t_1^{a_1}, \ldots, t_n^{a_n})^a$. For rules 6 and 7, use Theorem 6.7. □

Corollary 7.7 *If $\Gamma \vdash f : (t_1^{a_1}, \ldots, t_n^{a_n})^a$ and φ is a bijection $\{1, \ldots, n\} \to \{1, \ldots, n\}$ then there is a context Γ' and a pseudomatrix f' which is α-equal to f such that*

$$\Gamma' \vdash f' : (t_{\varphi(1)}^{a_{\varphi(1)}}, \ldots, t_{\varphi(n)}^{a_{\varphi(n)}})^a.$$

PROOF: By the second Free Variable Theorem, we can assume that $x_1 < \ldots < x_n$ are the free variables of f, and that $x_i{:}t_i^{a_i} \in \Gamma$ for all $i \in \{1, \ldots, n\}$.
Take n new free variables $z_1 < \ldots < z_n$ such that $z_i > y$ for all $y \in \mathrm{dom}(\Gamma)$.
Apply rule 7 n times. First on the $\varphi(1)$th free variable of f, resulting in

$$\Gamma_1 \vdash f_1 : (t_2^{a_2}, \ldots, t_{\varphi(1)-1}^{a_{\varphi(1)-1}}, t_{\varphi(1)+1}^{a_{\varphi(1)+1}}, \ldots, t_n^{a_n}, t_{\varphi(1)}^{a_{\varphi(1)}})^a.$$

Then on the $\varphi(2)$th free variable of f, and so on, finally resulting in

$$\Gamma_n \vdash f_n : (t_{\varphi(1)}^{a_{\varphi(1)}}, \ldots, t_{\varphi(n)}^{a_{\varphi(n)}})^a$$

Note that f_n is α-equal to f, as only variable substitutions have taken place. □

The following lemma shows that orders are superfluous as long as we restrict ourselves to pseudo-matrices (see also 3.4).

Lemma 7.8 *If* $\Gamma \vdash f : (t_1^{a_1}, \ldots, t_n^{a_n})^a$ *then* $a = 1 + \max(a_1, \ldots, a_n)$.

PROOF: Directly from the definition of $\Gamma \vdash f : (t_1^{a_1}, \ldots, t_n^{a_n})^a$. □

Corollary 7.9 (Unicity of type for pseudomatrices) *Assume,* Γ *is a context,* f *is a pseudo-matrix and* $\Gamma \vdash f : (t_1^{a_1}, \ldots, t_n^{a_n})^a$, *and* $\Gamma \vdash f : (u_1^{b_1}, \ldots, u_m^{b_m})^b$.
Then $(t_1^{a_1}, \ldots, t_n^{a_n})^a = (u_1^{b_1}, \ldots, u_m^{b_m})^b$.

PROOF: Immediate from Theorem 7.6 and Lemma 7.8. □

We can not leave out the context Γ in Corollary 7.9. For example, the pseudomatrix $\mathbf{z(x)}$ can have different types in different contexts, as is illustrated by the following derivations:

$$\dfrac{\dfrac{\vdash \mathtt{R(a_1)} : ()^0 \quad \vdash \mathtt{a_1} : 0^0}{\mathtt{x} : 0^0 \vdash \mathtt{R(x)} : (0^0)^1}\,3 \qquad \dfrac{\vdash \mathtt{R(a_1)} : ()^0 \quad \vdash \mathtt{a_1} : 0^0}{\mathtt{x} : 0^0 \vdash \mathtt{R(x)} : (0^0)^1}\,3}{\mathtt{x} : 0^0, \mathtt{z} : (0^0)^1 \vdash \mathtt{z(x)} : (0^0, (0^0)^1)^2}\,4$$

versus

$$\dfrac{\dfrac{\vdash \mathtt{R(a_1)} : ()^0 \quad \vdash \mathtt{R(a_1)} : ()^0}{\mathtt{x} : ()^0 \vdash \mathtt{x()} : (()^0)^1}\,4}{\mathtt{x} : ()^0, \mathtt{z} : (()^0)^1 \vdash \mathtt{z(x)} : (()^0, (()^0)^1)^2}\,4$$

Corollary 7.9 shows that our system RTT makes sense, in a certain way: The type of a pseudomatrix only depends on the context and does not depend on the way in which we derived the type of that pseudomatrix.

We feel that $\Gamma \vdash f : (t_1^{a_1}, \ldots, t_n^{a_n})^a$ tells us also something about the types (in the context Γ) of subtrees g of the pseudomatrix f, and their relation to the type of f. To give a simple example, if $k \in \mathcal{P}$, $z \in \mathcal{V}$ and $\Gamma \vdash z(k) : ((t^a)^{a+1})^{a+2}$, then we might expect that not only $z : (t^a)^{a+1} \in \Gamma$, but also that the argument k of z is of type t^a, so: $\Gamma \vdash k : t^a$.
This is worked out in the next theorem. For the proof of this theorem, we need an extension of the notion of recursive arguments (see Definition 2.11) to substitution trees, and the two lemmas below.

Definition 7.10 If $f\{x_i{:=}g_i\}$ is a substitution, then we define

$$\mathrm{RA}(f\{x_i{:=}g_i\}) \stackrel{\mathrm{def}}{=} \mathrm{RA}(f) \cup \{g_1, \ldots, g_n\} \cup \bigcup_{g_i \in \mathcal{P}} \mathrm{RA}(g_i)$$

For substitution trees $S \vee T$ we define $\mathrm{RA}(S \vee T) \stackrel{\mathrm{def}}{=} \mathrm{RA}(S) \cup \mathrm{RA}(T)$, and for substitution trees $\neg S$ we define $\mathrm{RA}(\neg S) \stackrel{\mathrm{def}}{=} \mathrm{RA}(S)$.

Note that if $f\{x_i{:=}g_i\}$ is an empty substitution, then $\mathrm{RA}(f\{x_i{:=}g_i\})$ according to the above definition is exactly $\mathrm{RA}(f)$, as defined in Definition 2.11. So the above definition agrees with Definition 2.11.

The following lemma states that a reduction doesn't create new recursive arguments.

Lemma 7.11 *Let S, T be substitution trees, and assume $S \longrightarrow T$.*
Then $\mathrm{RA}(T) \subseteq \mathrm{RA}(S)$.

PROOF: We use induction on the substitution tree S.

1. S is a substitution $f\{x_1{:=}g_1, \ldots, x_n{:=}g_n\}$.
   Use induction on the pseudomatrix f:
   (a) $f = R(i_1, \ldots, i_{a(R)})$. Then $T = R(i'_1, \ldots, i'_{a(R)})$, where the i'_j are defined as in Definition 4.6.1a.
   $\mathrm{RA}(T) = \{i'_1, \ldots, i'_{a(R)}\} \subseteq \{i_1, \ldots, i_{a(R)}, g_1, \ldots, g_n\} \subseteq \mathrm{RA}(S)$.
   (b) $f = z(k_1, \ldots, k_m)$. Compute $z'(k'_1, \ldots, k'_m)$ as described in Definition 4.6.1b.
   If S is easy then $\mathrm{RA}(T) = \mathrm{RA}(z'(k'_1, \ldots, k'_m))$ and from the definition of the k'_js it follows immediately that $\mathrm{RA}(T) \subseteq \mathrm{RA}(S)$.
   If S is hard, say: $z'(k'_1, \ldots, k'_m) \rightsquigarrow_2 g_q\{y_i{:=}k'_i\} = T$, then $\mathrm{RA}(T) = \mathrm{RA}(g_q) \cup \{k'_1, \ldots, k'_m\} \cup \bigcup_{k'_i \in \mathcal{P}} \mathrm{RA}(k'_i) \subseteq \mathrm{RA}(S)$.
   (c) $f = f_1 \vee f_2$. Then $T = T_1 \vee T_2$ where $f_j\{x_i{:=}g_i\} \longrightarrow T_j$ for $j = 1, 2$. By the induction hypothesis, $\mathrm{RA}(T_j) \subseteq \mathrm{RA}(f_j\{x_i{:=}g_i\})$, hence:

$$\begin{aligned} \mathrm{RA}(T) &= \mathrm{RA}(T_1) \cup \mathrm{RA}(T_2) \\ &\subseteq \mathrm{RA}(f_1\{x_i{:=}g_i\}) \cup \mathrm{RA}(f_2\{x_i{:=}g_i\}) \\ &= \mathrm{RA}(f\{x_i{:=}g_i\}) = \mathrm{RA}(S) \end{aligned}$$

   A similar argument holds if $f = \neg f'$.
2. $S = S_1 \vee S_2$, where S_1 and S_2 are substitution trees. Then $T = T_1 \vee S_2$, or $T = S_1 \vee T_2$, where $S_j \longrightarrow T_j$, assume: $T = T_1 \vee S_2$ (the other case is similar).
   By the induction hypothesis, $\mathrm{RA}(T_1) \subseteq \mathrm{RA}(S_1)$. A calculation similar to that in case 1(c) shows that $\mathrm{RA}(T) \subseteq \mathrm{RA}(S)$.
   The case $S = \neg S'$ is similar.

□

Corollary 7.12 *Let $f\{x_i{:=}g_i\}$ be a substitution. Then $\mathrm{RA}(f[x_i{:=}g_i]) \subseteq \mathrm{RA}(f\{x_i{:=}g_i\})$.*

PROOF: Induction on the length of the induction path from $f\{x_i{:=}g_i\}$ to $f[x_i{:=}g_i]$ with the help of the lemma above. □

If we have a derivation of the fact that f is a matrix in the context Γ, and $g \in \mathcal{P}$ is a recursive argument of f, then g is a matrix, too, as is proved by the following lemma and corollary.

Lemma 7.13 *Assume, we have a derivation of $\Gamma \vdash f : t^a$ for some pseudomatrix f. Let $g \in \mathcal{P} \cap \mathrm{RA}(f)$. Let Δ be the unique subset of Γ such that $\mathrm{dom}(\Delta)$ contains exactly all the variables of g.*
Then there is a context $\Delta' \supseteq \Delta$ and a type u^b such that $\Delta' \vdash g : u^b$ occurs in the derivation of $\Gamma \vdash f : t^a$.

PROOF: We use induction on the derivation of $\Gamma \vdash f : t^a$:

1. f is a basic formula. Then $\mathrm{RA}(f) \cap \mathcal{P} = \emptyset$ and there is nothing to prove.

2. The last derivation step was an application of rule 2 with main premises $\Gamma_i \vdash f_i : t_i^{a_i}$ $(i = 1, 2)$, and $f = f_1 \vee f_2$.
   Note: $g \in \mathrm{RA}(f_1) \cup \mathrm{RA}(f_2)$, say: $g \in \mathrm{RA}(f_i)$.
   By Lemma 3.16, $\Delta \subseteq \Gamma_i$, so the induction hypothesis gives Δ' and u^b such that $\Delta' \supseteq \Delta$ and $\Delta' \vdash g : u^b$ occurs in the derivation of $\Gamma_i \vdash f_i : t_i^{a_i}$.
   Then $\Delta' \vdash g : u^b$ also occurs in the derivation of $\Gamma \vdash f : t^a$.
   A similar argument holds for $f = \neg f'$.
3. The last step of the derivation used rule 3 with main premise $\Gamma' \vdash f' : t_1^{a_1}$.
   As g is a pseudomatrix, $g \in \mathrm{RA}(f')$, and by Lemma 3.16, $\Delta \subseteq \Gamma'$.
   With the induction hypothesis, we can determine $\Delta' \supseteq \Delta$ and u^b such that $\Delta' \vdash g : u^b$ occurs in the derivation of $\Gamma' \vdash f' : t_1^{a_1}$.
   Then $\Delta' \vdash g : u^b$ also occurs in the derivation of $\Gamma \vdash f : t^a$.
4. Use exactly the same argument as in 3.
5. Use Lemma 3.16 and the induction hypothesis.
6. The last derivation step used the substitution rule with main premise $\Gamma_1 \vdash f_1 : t_1^{a_1}$ and minor premise $\Gamma_2 \vdash f_2 : t_2^{a_2}$, say: $f = f_1[y{:=}f_2]$.
   f trivially occurs in the reduction path from $f_1\{y{:=}f_2\}$ to f, hence by Corollary 7.12, $g \in \mathrm{RA}(f_1)$, $g = f_2$ or $g \in \mathrm{RA}(f_2)$.
   Determine (in the first case: Use the induction hypothesis on $\Gamma_1 \vdash f_1 : t_1^{a_1}$, in the second case: take $\Delta' = \Gamma_2$ and $u^b = t_2^{a_2}$, in the third case: Use the induction hypothesis on $\Gamma_2 \vdash f_2 : t_2^{a_2}$) $\Delta' \supseteq \Delta$ and u^b such that $\Delta' \vdash g : u^b$ occurs in the derivation of $\Gamma_1 \vdash f_1 : t_1^{a_1}$ (in the first case) or $\Gamma_2 \vdash f_2 : t_2^{a_2}$ (in the second and the third case).
   In any case, $\Delta' \vdash g : u^b$ occurs in the derivation of $\Gamma \vdash f : t^a$.
7. Use Lemma 3.16 and the induction hypothesis.

□

Corollary 7.14 *If $\Gamma \vdash f : t^a$ and $g \in \mathcal{P} \cap \mathrm{RA}(f)$ then there is a unique type u^b such that $\Gamma \vdash g : u^b$.*

PROOF: Let Δ be the unique subset of Γ such that $\mathrm{dom}(\Delta)$ contains exactly the variables that occur in g.
Determine with Lemma 7.13 a context $\Delta' \supseteq \Delta$ and a type u^b such that $\Delta' \vdash g : u^b$ occurs in the derivation of $\Gamma \vdash f : t^a$.
By Lemma 3.16, $\Delta \vdash g : u^b$.
With the weakening rule: $\Gamma \vdash g : u^b$.
The uniqueness of u^b is a consequence of Corollary 7.9. □

Corollary 7.15 *Assume, we have a derivation of $\Gamma \vdash f : t^a$, and the last step of this derivation uses the substitution rule with main premise $\Gamma_1 \vdash f_1 : t_1^{a_1}$ and minor premise $\Gamma_2 \vdash f_2 : t_2^{a_2}$, say: $f = f_1[y{:=}f_2]$.*
Let $h\{y_i{:=}k_i\}$ be a substitution that occurs in one of the substitution trees of the reduction path from $f_1\{y{:=}f_2\}$ to f, and let $g \in \mathcal{P}$.
If $g \in \mathrm{RA}(h\{y_i{:=}k_i\})$ and $\Delta \subseteq \Gamma_1 \cup \Gamma$ is the context that contains exactly the variables that occur in g, then there is $\Delta' \supseteq \Delta$ and u^b such that $\Delta' \vdash g : u^b$ occurs in the derivation of $\Gamma \vdash f : t^a$.

PROOF: Directly from Lemma 7.11 and Lemma 7.13 □

Theorem 7.16 *Let Γ be a context, and f a pseudomatrix such that $\Gamma \vdash f : t_0^{a_0}$. Let g be a simple submatrix of f.*

(*) *If $g = R(i_1, \ldots, i_{\mathfrak{a}(R)})$ then $i_j : \mathsf{0}^{\mathsf{0}} \in \Gamma$ or $\Gamma \vdash i_j : \mathsf{0}^{\mathsf{0}}$ for all $j \leq \mathfrak{a}(R)$;*

(**) *If $g = z(k_1, \ldots, k_m)$ then there are $a, t_1^{a_1}, \ldots, t_m^{a_m}$ such that $z : (t_1^{a_1}, \ldots, t_m^{a_m})^a \in \Gamma$ and $k_j : t_j^{a_j} \in \Gamma$ or $\Gamma \vdash k_j : t_j^{a_j}$ for all $j \leq m$.*

PROOF: We use induction on $\Gamma \vdash f : t_0^{a_0}$.

1. Assume f is a basic formula. Then $g = f$. It is clear that we are in case $(*)$ and that $\Gamma \vdash i_j : 0^0$ for all $j \leq \mathfrak{a}(R)$.
2. Assume, $\Gamma = \Gamma_1 \cup \Gamma_2$, and $f = f_1 \vee f_2$, and $\Gamma \vdash f : t_0^{a_0}$ is derived by means of rule 2 from $\Gamma_1 \vdash f_1 : t_{01}^{a_{01}}$ and $\Gamma_2 \vdash f_2 : t_{02}^{a_{02}}$.
   As g is simple, g must be a submatrix of f_1 or f_2, say, f_p.
   $(*)$ $g = R(i_1, \ldots i_{\mathfrak{a}(R)})$. Note: $\Gamma_p \subseteq \Gamma$, so:
   $$i_j : 0^0 \in \Gamma_p \Rightarrow i_j : 0^0 \in \Gamma$$
   and, with the use of rule 5 (weakening):
   $$\Gamma_p \vdash i_j : 0^0 \Rightarrow \Gamma \vdash i_j : 0^0,$$
   so we are done by the induction hypothesis on $\Gamma_p \vdash f_p : t_{0p}^{a_{0p}}$.
   $(**)$ $g = z(k_1, \ldots, k_m)$. Use the induction hypothesis, as in case $(*)$.
   A similar argument holds in case $f = \neg f'$.
3. Assume $\Gamma \vdash f : t_0^{a_0}$ has been derived with rule 3 from the main premise $\Gamma' \vdash h : t_{01}^{a_{01}}$ and the minor premise $\Gamma' \vdash k : t_{02}^{a_{02}}$.
   Let g' be the submatrix of h that is transformed into g when we apply rule 3 on h and k.
   For the arguments of g that are also present in g', the result follows from the induction hypothesis. If k'' is an argument that was not present in g', then it must be the newly introduced variable. It replaces an argument k' of g'.
   Note that k' must be α_Γ-equal to k, according to rule 3. By Corollary 7.14, we know that k' is a matrix in the context Γ, and by Theorem 7.6 and Corollary 7.9, we see that the type of k' must be equal to the type of k.
   As the newly introduced variable also has the same type as k, the types of k' and k'' are the same, and we are done by the induction hypothesis.
4. Assume $\Gamma \vdash f : t_0^{a_0}$ has been derived with rule 4 from the main premise $\Gamma' \vdash h : t_{01}^{a_{01}}$ and the minor premise $\Gamma' \vdash k : t_{02}^{a_{02}}$.
   Let g' be the submatrix of h that is transformed into g when we apply rule 4 on h and k.
   If $g = g'$ then we are done by the induction hypothesis.
   If $g \neq g'$ then $g = z(x_1, \ldots, x_m)$, where z is the newly introduced variable and $x_1 < \ldots < x_m$ are the free variables of g'. $x_1, \ldots, x_m$ are in the domain of Γ', assume: $x_i : t_i^{a_i} \in \Gamma'$.
   Let $y_1 < \ldots < y_n$ be the free variables of k.
   As g' and k are $\alpha_{\Gamma'}$-equal (by definition: see rule 4 of RTT), $m = n$, and $y_i : t_i^{a_i} \in \Gamma'$ for all i.
   As the y_i occur in g' and therefore in f, $y_i : t_i^{a_i} \in \Gamma$ for all i.
   By Theorem 7.6 and Lemma 7.8, k must have type $(t_1^{a_1}, \ldots, t_m^{a_m})^a$ in the context Γ'. Here, $a = 1 + \max(a_1, \ldots, a_m)$.
   Therefore, the newly introduced variable z has type $(t_1^{a_1}, \ldots, t_m^{a_m})^a$ in the context Γ. As the y_is have type $t_i^{a_i}$ in Γ, the results of $(**)$ are clear now.
5. Trivial.
6. Assume, $\Gamma \vdash f : t_0^{a_0}$ has been derived with rule 6 from the major premise $\Gamma' \vdash f' : t_{01}^{a_{01}}$ and the minor premise $\Gamma'' \vdash k : t_{02}^{a_{02}}$, say: $f = f'[y{:=}k]$.
   Let $S_0, S_1, \ldots, S_p$ be the reduction path from $f'\{y{:=}k\}$ to $f'[y{:=}k]$. This path is well-typed.
   Let $h\{y_i{:=}g_i\}$ be a substitution that occurs in one of the S_j. We prove $(*)$ and $(**)$ for simple submatrices g of $h, g_1, \ldots, g_n$ (as far as the g_is are pseudomatrices, of course). We do this by induction on j.
   - $j = 0$: Then $h\{y_i{:=}g_i\} = f'\{y{:=}k\}$ and the results follow from the induction hypotheses on f' and k.
   - Assume we have proved the theorem for S_j. Use induction on the substitution tree S_j to prove the result for S_{j+1}:
     (a) S_j is a substitution $h'\{x_i{:=}k_i'\}$.

i. $h' = R(i_1, \ldots, i_{a(R)})$. Then $S_{j+1} = h = h'[x_i{:=}k'_i]$, and $g = h$.
As the substitution is well-typed, all the arguments of g are of type 0^0, because those of h' are, by the induction hypothesis.

ii. $h' = z(k_1, \ldots, k_m)$. If the substitution is easy, then hold an argument similar to the case $h' = R(i_1, \ldots, i_{a(R)})$.
Now assume the substitution is hard. Note that $h = k'_i$ for some i, and as we have proved the result already for simple submatrices of k'_i, we are done for h.
If g is a simple submatrix of g_q then either $g_q = k'_i$ and we are done again, or $g_q = k_i$. In the last case, use Corollary 7.15: There are contexts Δ' and Δ such that $\mathrm{dom}(\Delta)$ contains exactly the variables of g_q, $\Delta \subseteq \Delta'$, $\Delta' \vdash g_q : u^b$ occurs in the derivation of $\Gamma \vdash f : t_0^{a_0}$, $\Delta \subseteq \Gamma$.
Use the induction hypothesis on $\Delta' \vdash g_q : u^b$ to obtain the desired result.

iii. $h' = h_1 \vee h_2$. Then $S_j \longrightarrow T_1 \vee T_2$, where $h_1\{x_i{:=}k'_i\} \longrightarrow T_1$ and $h_2\{x_i{:=}k'_i\} \longrightarrow T_2$. The result follows from the results for $h_j\{x_i{:=}k'_i\} \longrightarrow T_j$. A similar argument holds if $h' = \neg h''$.

(b) S_j is a substitution tree. Use a similar argument as in (a)iii.

7. Directly from the induction hypothesis.

□

With the theorems 7.6 and 7.16 we can prove that a well-typed substitution always can be carried out:

Theorem 7.17 *If S is a well-typed substitution tree in the context Γ then there is a well-typed substitution tree T such that $S \longrightarrow_\Gamma T$, or there is a pseudomatrix f such that $S \longrightarrow_\Gamma f$.*

PROOF:

1. S is a substitution: $S = f\{x_1{:=}g_1, \ldots, x_n{:=}g_n\}$.

   (a) If $f = R(i_1, \ldots, i_{a(R)})$ then note that each x_i that occurs free in f has type 0^0 (7.16), and that therefore $S \longrightarrow R(i'_1, \ldots, i'_{a(R)})$ as described in 4.6.1(a).

   (b) If $f = z(k_1, \ldots, k_m)$ and S is easy then note that z can not have type 0^0 (7.16). So if $z = x_i$ then $g_i \notin \mathcal{A}$ because of the well-typedness of the substitution, and therefore $S \longrightarrow z'(k'_1, \ldots, k'_m)$ as described in 4.6.1(b).
   If S is hard then determine $z'(k'_1, \ldots, k'_m)$ as described in 4.6.1(b). z' must have the same type as z because the substitution is well-typed. According to Theorem 7.16, there are $t_1^{a_1}, \ldots, t_m^{a_m}, a$ such that $z : (t_1^{a_1}, \ldots, t_m^{a_m})^a \in \Gamma$, hence:

   $$\Gamma \vdash z' : (t_1^{a_1}, \ldots, t_m^{a_m})^a.$$

   Theorem 7.6 assures that z' has exactly m variables, say, $y_1 < \ldots < y_m$, and that $y_i : t_i^{a_i} \in \Gamma$. Moreover, Theorem 7.16 assures that k'_i also has type $t_i^{a_i}$ in the context Γ. Finally, we need to check that for any submatrix $g = z''(k''_1, \ldots, k''_q)$ of z', $z'' = y_i$ and $k''_r = y_j$ implies $k'_i \neq k'_j$. Apply Theorem 7.16(**) on $\Gamma \vdash z' : (t_1^{a_1}, \ldots, t_n^{a_n})^a$ and g. This yields that z'' and k''_r have different types in Γ. Therefore, the types of k'_i and k'_j are different, too. By unicity of types, k'_i and k'_j can not be equal.
   Hence $z'\{y_i{:=}k'_i\}$ is a well-typed substitution.

   (c) If $f = f_1 \vee f_2$ and we know already that $f_1\{x_i{:=}g_i\}$ and $f_2\{x_i{:=}g_i\}$ reduce to pseudomatrices or well-typed substitution trees (say, T_1 and T_2), then it is easy to see that $S \longrightarrow_\Gamma T_1 \vee T_2$, and that $T_1 \vee T_2$ is a well-typed substitution tree.
   A similar argument holds for the case $f = \neg f'$.

2. If $S = T_1 \vee T_2$, where T_1 and T_2 are substitution trees, a similar argument as in 1(c) shows that S reduces to a well-typed substitution tree.

□

Now we are ready to fill up the last important gap in the definition of substitution in RTT:

Corollary 7.18 *Assume* $\Gamma \cup \{y : t_i^{a_i}\} \vdash f : t^a$ *and* $\Gamma \vdash k : t_i^{a_i}$ *are the main and minor premise of rule 3.9.6. Then there is always a pseudomatrix* h *such that* $h = f[y{:=}k]_{\Gamma \cup \{y:t_i^{a_i}\}}$.

PROOF: $f\{y{:=}k\}$ is a well-typed substitution in the context $\Gamma \cup \{y : t_i^{a_i}\}$. □

8 Matrices in the Principia Mathematica

We wonder whether our definition of matrix (see 3.10) coincides with the definition of matrix that was given in the Principia. To answer this, we need a clear view on which pseudomatrices are matrices in RTT, and which are not. We develop this view in some lemmas below.

We do not distinguish between pseudomatrices that are α-equal, nor between types $(t_1, \ldots, t_n)$ and $(t_{\varphi(1)}, \ldots, t_{\varphi(n)})$.
This is justified by Corollary 7.7 and by the fact, that matrices that are α-equal are supposed to be the same in the Principia, too.
We define the notion "up to α-equality" formally:

Definition 8.1 Let $f \in \mathcal{P}$, Γ a context, t^a a type. f is of type t^a in the context Γ *up to α-equality*, notation $\Gamma \vdash f : t^a (\mathrm{mod}\ \alpha)$ if there is $f' \in \mathcal{P}$, a context Γ' and a bijection $\varphi : \mathcal{V} \to \mathcal{V}$ such that

- $\Gamma' \vdash f' : t^a$;
- f' is the pseudomatrix resulting from f if each variable x is replaced by $\varphi(x)$ (so: f' and f are α-equal via the bijection φ);
- $x < y \Rightarrow \varphi(x) < \varphi(y)$ for all $x, y \in \mathrm{dom}(\Gamma)$;
- $\Gamma' = \{\varphi(x){:}u^b | x{:}u^b \in \Gamma\}$.

We say that f is a matrix in the context Γ *up to α-equality* if there is a type u^b such that $\Gamma \vdash f : u^b (\mathrm{mod}\ \alpha)$.
We say that f is a matrix *up to α-equality* if there is a context Γ such that f is a matrix in Γ up to α-equality.

Definition 8.2 We define the notion of *free type*:

1. 0^0 is a free type;
2. If $t_1^{a_1}, \ldots, t_n^{a_n}$ are free types, then $(t_1^{a_1}, \ldots, t_n^{a_n})^a$ is a free type. Here, $a = \max(a_1, \ldots, a_n)+1$; if $n = 0$ then we take $a = 0$.
3. All free types can be constructed using the construction rules 1 and 2 above.

We have an important extension of Lemma 7.8:

Lemma 8.3 *If* $\Gamma \vdash f : t^a$ *then* t^a *is a free type.*

PROOF: Induction on $\Gamma \vdash f : t^a$. □

Lemma 8.4 *For all free types* t^a *there are* f *and* Γ *such that* $\Gamma \vdash f : t^a$.

PROOF: We use induction on free types. For the sake of clarity, we leave out the orders in this proof. This is justified by Remark 3.12.
The case $t = 0$ is trivial.
Now assume $t = (t_1, \ldots, t_m)$, and for all $i \leq m$:

$$\Gamma_i \vdash f_i : t_i$$

and $x_{i,1} < \ldots < x_{i,m_i}$ are the free variables of f_i.
Take variables $z_1 < \ldots < z_m$ such that $x < z_i$ for all $x \in \mathrm{dom}(\Gamma_i)$.
Distinguish 2 cases:

- $t_i = \mathtt{0}$. Then make the following derivation:

$$\frac{\Gamma_i \vdash \mathtt{R}(f_i) : () \qquad \Gamma_i \vdash f_i : \mathtt{0}}{\Gamma_i, z_i{:}\mathtt{0} \vdash \mathtt{R}(z_i) : (\mathtt{0})}\,3$$

  Write $\Delta_i = \Gamma_i \cup \{z_i{:}\mathtt{0}\}$, and $g_i = \mathtt{R}(z_i)$.

- $t_i \neq \mathtt{0}$. Because of Theorem 7.6, there are types $u_{i,1}, \ldots, u_{i,m_i}$ such that $x_{i,j} : u_{i,j} \in \Gamma_i$ and $t_i = (u_{i,1}, \ldots, u_{i,m_i})$.
  Now use rule 4 of RTT:

$$\frac{\Gamma_i \vdash f_i : t_i \qquad \Gamma_i \vdash f_i : t_i}{\Gamma_i, z_i{:}t_i \vdash z_i(x_{i,1}, \ldots, x_{i,m_i}) : (u_{i,1}, \ldots, u_{i,m_i}, t_i)}\,4$$

  Use the induction hypothesis on t_i and determine k_j and $\Gamma_{i,j}$ such that $\Gamma_{i,j} \vdash k_j : u_{i,j}$ for all $j \leq m_i$.
  We can assume that the domains of the contexts $\{z_i{:}t_i\}, \Gamma_i, \Gamma_{i,1}, \ldots, \Gamma_{i,m_i}$ are pairwise disjoint. Let Δ_i be the union of these contexts.
  Now we apply rule 6 m_i times (sometimes we need to use the weakening rule before and after we apply this rule, but we do not write this explicitly here):
  First, on $k_{i,1}$:

$$\frac{\Delta_i \vdash z_i(x_{i,1}, \ldots, x_{i,m_i}) : (u_{i,1}, \ldots, u_{i,m_i}, t_i) \qquad \Delta_i \setminus \{x_{i,1}{:}u_{i,1}\} \vdash k_{i,1} : u_{i,1}}{\Delta_i \vdash z_i(k_{i,1}, x_{i,2}, \ldots, x_{i,m_i}) : (u_{i,2}, \ldots, u_{i,m_i}, t_i)}\,6$$

  Then on $k_{i,2}$:

$$\frac{\Delta_i \vdash z_i(k_{i,1}, x_{i,2}, \ldots, x_{i,m_i}) : (u_{i,2}, \ldots, u_{i,m_i}, t_i) \qquad \Delta_i \setminus \{x_{i,2}{:}u_{i,2}\} \vdash k_{i,2} : u_{i,2}}{\Delta_i \vdash z_i(k_{i,1}, k_{i,2}, x_{i,3}, \ldots, x_{i,m_i}) : (u_{i,3}, \ldots, u_{i,m_i}, t_i)}\,6$$

  and so on. Finally, we reach:

$$\Delta_i \vdash z_i(k_{i,1}, \ldots, k_{i,m_i}) : (t_i)$$

  Write $g_i = z_i(k_{i,1}, \ldots, k_{i,m_i})$.

For $i = 1, \ldots, m$ we now have:

$$\Delta_i \vdash g_i : (t_i).$$

We can assume that $x < y$ for $x \in \mathrm{dom}(\Delta_i)$, $y \in \mathrm{dom}(\Delta_j)$ with $i < j$.
Write $\Delta = \Delta_1 \cup \ldots \cup \Delta_m$.
Apply rule 2 $(m{-}1)$ times to obtain:

$$\Delta \vdash g_1 \vee (g_2 \vee \ldots \vee (g_{m-1} \vee g_m) \ldots) : (t_1, \ldots, t_m)$$

□

Corollary 8.5 *t^a is a free type if and only if there is a context Γ and $f \in \mathcal{A} \cup \mathcal{P}$ such that $\Gamma \vdash f : t^a$.*

PROOF: From Lemma 8.3 and Lemma 8.4. □

Example 8.6 *We find a pseudomatrix f and a context Γ such that $\Gamma \vdash f : (\mathtt{0}^{\mathtt{0}}, \mathtt{0}^{\mathtt{0}}, ((\mathtt{0}^{\mathtt{0}})^{\mathtt{1}})^{\mathtt{2}})^{\mathtt{3}}$. For reasons of clarity and space, we leave out the orders, again. We follow the proof of Lemma 8.4:*

- *First of all, we have to find contexts $\Gamma_1, \Gamma_2, \Gamma_3$ and pseudomatrices f_1, f_2, f_3 such that $\Gamma_1 \vdash f_1 : \mathtt{0}$, $\Gamma_2 \vdash f_2 : \mathtt{0}$ and $\Gamma_3 \vdash f_3 : ((\mathtt{0}))$:*

  1. *Take $\Gamma_1 = \emptyset$, $f_1 = \mathtt{a}_{\mathtt{1}}$;*
  2. *Take $\Gamma_2 = \emptyset$; $f_2 = \mathtt{a}_{\mathtt{1}}$;*

3. *To find Γ_3 and f_3, we need to carry out the proof of Lemma 8.4 on the type $((0))$. The reader is invited to check that we can take $\Gamma_3 = \{z_1 : (0)\}$ and $f_3 = z_1(a_1)$.*

- *We take three new variables, x_1 for $\Gamma_1 \vdash f_1 : 0$, y_1 for $\Gamma_2 \vdash f_2 : 0$ and z_2 for $\Gamma_3 \vdash f_3 : ((0))$.*

1. *We derive for f_1:*

$$\frac{\vdash R(a_1) : () \qquad \vdash a_1 : 0}{x_1 : 0 \vdash R(x_1) : (0)}\,3$$

2. *We derive for f_2:*

$$\frac{\vdash R(a_1) : () \qquad \vdash a_1 : 0}{y_1 : 0 \vdash R(y_1) : (0)}\,3$$

3. *We derive for f_3:*

$$\frac{\dfrac{\dfrac{z_1{:}(0) \vdash z_1(a_1) : ((0))}{z_1{:}(0), z_2{:}((0)) \vdash z_2(z_1) : ((0),((0)))}\,4}{z_1{:}(0), z_2{:}((0)), z_3{:}0 \vdash z_2(z_1) : ((0),((0)))}\,5 \qquad \dfrac{\dfrac{\vdash R(a_1) : () \quad \vdash a_1 : 0}{z_3{:}0 \vdash R(z_3) : (0)}\,3}{z_2{:}((0)), z_3{:}0 \vdash R(z_3) : (0)}\,5}{z_2{:}((0)), z_3{:}0 \vdash z_2(R(z_3)) : (((0)))}\,6$$

- *Call the conclusions of these three derivations: $\Delta_1 \vdash g_1 : (0)$, $\Delta_2 \vdash g_2 : (0)$ and $\Delta_3 \vdash g_3 : (((0)))$.*
  Finally, we apply rule 2 twice:

$$\frac{\dfrac{\Delta_1 \vdash g_1 : (0) \qquad \Delta_2 \vdash g_2 : (0)}{\Delta_1 \cup \Delta_2 \vdash g_1 \vee g_2 : (0,0)}\,2 \qquad \Delta_3 \vdash g_3 : (((0)))}{x_1{:}0, y_1{:}0, z_2{:}((0)), z_3{:}0 \vdash (R(x_1) \vee R(y_1)) \vee z_2(R(z_3)) : (0,0,((0)))}\,2$$

and we have the desired derivation.

Lemma 8.7 *Assume $z \in \mathcal{V}$ and $t^a = (t_1^{a_1}, \ldots, t_n^{a_n})^a$ is a free type.*
Moreover, assume $x_1 < \ldots < x_n$ are distinct variables, and $z > x_i$ for all i.
Then $\{z : t^a\} \cup \{x_i : t_i^{a_i} | 1 \leq i \leq n\} \vdash z(x_1, \ldots, x_n) : (t_1^{a_1}, \ldots, t_n^{a_n}, t^a)^{a+1} (\mathrm{mod}\ \alpha)$.

PROOF: Use Lemma 8.4 to determine f and Γ such that $\Gamma \vdash f : t^a$.
As we are only interested in the derivability of $z(x_1, \ldots, x_n)$ up to α-equality, we may assume that $\mathrm{FV}(f) = \{x_1, \ldots, x_n\}$ and that $x_i : t_i^{a_i} \in \Gamma$ for all $i \leq n$.
Now use rule 4 of RTT to deduce

$$\{z{:}t^a\} \cup \{x_i{:}t_i^{a_i} | 1{\leq}i{\leq}n\} \vdash z(x_1, \ldots, x_n) : (t_1^{a_1}, \ldots, t_n^{a_n}, t^a)^{a+1}.$$

□

Lemma 8.8 *If $k_1, \ldots, k_n \in \mathcal{A} \cup \mathcal{V} \cup \mathcal{P}$, $t^a = (t_1^{a_1}, \ldots, t_n^{a_n})^a$ is a free type, $\Gamma \vdash k_i : t_i^{a_i}$ for all $k_i \in \mathcal{A} \cup \mathcal{P}$ and $k_i : t_i^{a_i} \in \Gamma$ for all $k_i \in \mathcal{V}$, and $z \in \mathcal{V} \setminus \mathrm{dom}(\Gamma)$, then $z(k_1, \ldots, k_n)$ is a matrix in the context $\Gamma \cup \{z : t^a\}$ (up to α-equality).*

PROOF: With the use of Lemma 8.7, we make a derivation of

$$\{z{:}t^a\} \cup \{x_i{:}t_i^{a_i} | q{\leq}i{\leq}n\} \vdash z(x_1, \ldots, x_n) : (t_1^{a_1}, \ldots, t_n^{a_n}, t^a)^{a+1} (\mathrm{mod}\ \alpha).$$

Find (with Lemma 8.4) $k'_1, \ldots, k'_m$ such that

- $k'_i = k_i$ if $k_i \in \mathcal{A} \cup \mathcal{P}$;
- $k'_i \in \mathcal{A} \cup \mathcal{P}$ has type $t_i^{a_i}$ in a context Δ_i if $k_i \in \mathcal{V}$;
- Δ, the union of the contexts Γ and the Δ_is, is a context.
- For $k_i \in \mathcal{V}$: $k'_i =_{\alpha,\Gamma} k'_j$ if and only if $k_i = k_j$.[23]

[23] One might wonder whether there are enough pseudomatrices of one type that are not α_Γ-equal. Lemma 8.4 provides only one pseudomatrix for each type. But if we have that pseudomatrix, say k, then we use rule 2 of RTT to create $\neg k$, $\neg\neg k$, $\neg\neg\neg k$, etc. k, $\neg k$, $\neg\neg k$,... are all α_Γ-different and of the same type as k.

Apply rule 6 m times (as in the proof of Lemma 8.4), and where necessary the weakening rule, to obtain:

$$\Delta \vdash z(k'_1, \ldots, k'_m) : (t^a)^{a+1} (\text{mod } \alpha).$$

Now introduce, with rule 3, new variables for the k'_i which are not equal to k_i, to obtain a matrix that is α-equal to $z(k_1, \ldots, k_m)$. □

Example 8.9 *Let* $\Gamma = \{\mathtt{x}{:}0, \mathtt{y}{:}0\}$, $k_1 = k_2 = \mathtt{x}$, $k_3 = \mathtt{y}$ *and* $k_4 = \mathtt{a}_1$. *We show that* $\mathtt{z}(k_1, k_2, k_3, k_4)$ *is a matrix in the context* $\Gamma \cup \{\mathtt{z}{:}(0,0,0,0)\}$ *(up to* α*-equality).*
We use Lemma 8.7 to make a derivation of

$$\{\mathtt{x}_1{:}0, \mathtt{x}_2{:}0, \mathtt{x}_3{:}0, \mathtt{x}_4{:}0, \mathtt{z}{:}(0,0,0,0)\} \vdash \mathtt{z}(\mathtt{x}_1, \mathtt{x}_2, \mathtt{x}_3, \mathtt{x}_4) : (0,0,0,0,(0,0,0,0)).$$

We define $k'_1 = \mathtt{a}_2$, $k'_2 = \mathtt{a}_2$, $k'_3 = \mathtt{a}_3$ *and* $k'_4 = \mathtt{a}_1$, *and:* $\Delta = \{\mathtt{z} : (0,0,0,0), \mathtt{x}_1 : 0, \mathtt{x}_2 : 0, \mathtt{x}_3 : 0, \mathtt{x}_4 : 0\}$.
Now we apply rule 6 four times (after each application, we must use the weakening rule. We do not write this down in the picture below):

$$\dfrac{\dfrac{\dfrac{\Delta \vdash \mathtt{z}(\mathtt{x}_1, \mathtt{x}_2, \mathtt{x}_3, \mathtt{x}_4) : (0,0,0,0,(0,0,0,0)) \qquad \Delta \vdash \mathtt{a}_2 : 0}{\Delta \vdash \mathtt{z}(\mathtt{a}_2, \mathtt{x}_2, \mathtt{x}_3, \mathtt{x}_4) : (0,0,0,(0,0,0,0)) \qquad \Delta \vdash \mathtt{a}_2 : 0}}{\Delta \vdash \mathtt{z}(\mathtt{a}_2, \mathtt{a}_2, \mathtt{x}_3, \mathtt{x}_4) : (0,0,(0,0,0,0))}}{\vdots}}{\Delta \vdash \mathtt{z}(\mathtt{a}_2, \mathtt{a}_2, \mathtt{a}_3, \mathtt{a}_1) : ((0,0,0,0))}$$

Finally, we re-introduce some variables with rule 3:

$$\dfrac{\dfrac{\Delta \vdash \mathtt{z}(\mathtt{a}_2, \mathtt{a}_2, \mathtt{a}_3, \mathtt{a}_1) : ((0,0,0,0)) \qquad \Delta \vdash \mathtt{a}_2 : 0}{\Delta \cup \{\mathtt{z}_1{:}0\} \vdash \mathtt{z}(\mathtt{z}_1, \mathtt{z}_1, \mathtt{a}_3, \mathtt{a}_1) : ((0,0,0,0),0) \qquad \Delta \cup \{\mathtt{z}_1{:}0\} \vdash \mathtt{a}_3 : 0}}{\Delta \cup \{\mathtt{z}_1{:}0, \mathtt{z}_2{:}0\} \vdash \mathtt{z}(\mathtt{z}_1, \mathtt{z}_1, \mathtt{z}_2, \mathtt{a}_1) : ((0,0,0,0),0,0)}$$

To obtain a result of type $(0,0,(0,0,0,0))$, *we apply rule 7 on the variable* $\mathtt{z}$.

Lemma 8.10 *If* f *and* g *are pseudomatrices,* $\Gamma_1 \vdash f : (t_1^{a_1}, \ldots, t_n^{a_n})^a$, $\Gamma_2 \vdash g : (u_1^{b_1}, \ldots, u_m^{b_m})^b$, *and* $\Gamma_1 \cup \Gamma_2$ *is a context, then* $f \vee g$ *is a matrix in the context* $\Gamma_1 \cup \Gamma_2$ *(up to* α*-equality).*

PROOF: For reasons of clarity, we again leave out the orders of the ramified types. We can not simply apply rule 2 of RTT, as the contexts Γ_1 and Γ_2 may not obey to the condition on them in rule 2. To solve this problem, we make a little roundabout.
Lemma 8.7 makes it possible to derive

$$\Delta_1 \vdash z_1(x_1, \ldots, x_n) : (t_1, \ldots, t_n, (t_1, \ldots, t_n)) (\text{mod } (\alpha)$$

$$\Delta_2 \vdash z_2(y_1, \ldots, y_m) : (u_1, \ldots, u_m, (u_1, \ldots, u_m)) (\text{mod } (\alpha)$$

in suitable contexts Δ_1, Δ_2, such that $\Delta_1 \cup \Delta_2$ is a context.
We can assume that $\{x_1, \ldots, x_n\} = \text{FV}(f)$ and $\{y_1, \ldots, y_m\} = \text{FV}(g)$ and $z_1 \neq z_2$.
Determine (with Lemma 8.4) a context Δ and for each $x_i \in \text{FV}(f) \cap \text{FV}(g)$ pseudomatrices k_i, k'_i such that

- $\Delta \vdash k_i : t_i^{a_i}$;
- $\Delta \vdash k'_i : t_i^{a_i}$;
- $k'_i =_{\alpha,\Delta} k_i$;

- if x is a variable in k_i and x' is a variable in k'_i then $x < x'$;[24]
- $\Delta_1 \cup \Delta_2 \subseteq \Delta$.

Use rule 6 to substitute these k_is for x_i in $z_1(x_1, \ldots, x_n)$ and also for x_i in $z_2(y_1, \ldots, y_m)$, resulting in matrices f' and g'.
With some appropriate applications of rule 7, we can obtain: If x is a variable of f', and x' is a variable of g', then $x < x'$.
Lemma 3.16 allows us to assume that f' was derived in a context Δ'_1 that has exactly the variables of f' as domain, and that g' was derived in a context Δ'_2 that has exactly the variables of g' as domain.
Now we can apply rule 2 and form $\Delta'_1 \cup \Delta'_2 \vdash f' \vee g' : v$ for a certain type v.
With rule 3, we replace the k_is and k'_is that were introduced before the application of rule 2, by variables.
Upto α-equality, we have now a derivation of $z_1(x_1, \ldots, x_m) \vee z_2(y_1, \ldots, y_n)$ in the context $\Delta_1 \cup \Delta_2$.
Now we substitute f for z_1 and g for z_2 and we obtain a derivation of $f' \vee g'$, as desired. □

Example 8.11 *From* $\{\mathtt{x}{:}(0), \mathtt{y}{:}((0))\} \vdash \mathtt{y}(\mathtt{x}) : ((0), ((0)))$ *and* $\{\mathtt{x}{:}(0), \mathtt{z}{:}0\} \vdash \mathtt{x}(\mathtt{z}) : ((0), 0)$ *we make a derivation of* $\{\mathtt{x}{:}(0), \mathtt{y}{:}((0)), \mathtt{z}{:}0\} \vdash \mathtt{y}(\mathtt{x}) \vee \mathtt{x}(\mathtt{z}) : ((0), 0, ((0)))$.
With Lemma 8.7, we derive

$$\{\mathtt{x}{:}(0), \mathtt{y}{:}((0)), \mathtt{y}_1{:}((0), ((0)))\} \vdash \mathtt{y}_1(\mathtt{x}, \mathtt{y}) : ((0), ((0)), ((0), ((0))))$$

$$\{\mathtt{x}{:}(0), \mathtt{z}{:}0, \mathtt{z}_1{:}((0), 0)\} \vdash \mathtt{z}_1(\mathtt{x}, \mathtt{z}) : ((0), 0, ((0), 0))$$

The only "problematic" variable is $\mathtt{x}$, *so we substitute something of type* (0) *for* $\mathtt{x}$.
We extend our contexts with $\mathtt{y}_2{:}0$ *and* $\mathtt{z}_2{:}0$ *respectively.*
In $\mathtt{y}_1(\mathtt{x}, \mathtt{y})$, *we substitute* $\mathtt{R}(\mathtt{y}_2)$ *for* $\mathtt{x}$ *and in* $\mathtt{z}_1(\mathtt{x}, \mathtt{z})$ *we substitute* $\mathtt{R}(\mathtt{z}_2)$ *for* $\mathtt{x}$. *At that point, we can apply rule 2:*

$$\frac{\{\mathtt{y}{:}((0)), \mathtt{y}_1{:}((0), ((0))), \mathtt{y}_2{:}0\} \vdash \mathtt{y}_1(\mathtt{R}(\mathtt{y}_2), \mathtt{y}) : (((0)), ((0), ((0)))) \qquad \{\mathtt{z}{:}0, \mathtt{z}_1{:}((0), 0), \mathtt{z}_2{:}0\} \vdash \mathtt{z}_1(\mathtt{R}(\mathtt{z}_2), \mathtt{z}) : (0, ((0), 0), 0)}{\{\mathtt{y}{:}((0)), \mathtt{y}_1{:}((0), ((0))), \mathtt{y}_2{:}0, \mathtt{z}{:}0, \mathtt{z}_1{:}((0), 0), \mathtt{z}_2{:}0\} \vdash \mathtt{y}_1(\mathtt{R}(\mathtt{y}_2), \mathtt{y}) \vee \mathtt{z}_1(\mathtt{R}(\mathtt{z}_2), \mathtt{z}) : (((0)), ((0), ((0))), 0, 0, ((0), 0), 0)}$$

Then we re-introduce the variable $\mathtt{x}$. *We cannot use the name* $\mathtt{x}$ *for it, but use* $\mathtt{z}_3$ *instead. We obtain the matrix*

$$\mathtt{y}_1(\mathtt{z}_3, \mathtt{y}) \vee \mathtt{z}_1(\mathtt{z}_3, \mathtt{z}).$$

Finally, we substitute $\mathtt{y}(\mathtt{x})$ *for* $\mathtt{y}_1$, *which results in*

$$\mathtt{y}(\mathtt{z}_3) \vee \mathtt{z}_1(\mathtt{z}_3, \mathtt{z}),$$

and $\mathtt{x}(\mathtt{z})$ *for* $\mathtt{z}_1$, *and obtain our final result*

$$\mathtt{y}(\mathtt{z}_3) \vee \mathtt{z}_3(\mathtt{z}).$$

This is a result of type $(((0)), 0, (0))$.
We obtain a result of type $((0), ((0)), 0)$ *if we apply rule 7 two times: first on the variable* $\mathtt{y}$, *and then on the variable* $\mathtt{z}$.

Theorem 8.12 *Let* f *be a pseudomatrix.* f *is a matrix (up to* α*-equality) if and only if*

1. $f = R(i_1, \ldots, i_{a(R)})$, *or*

[24] It is not hard to see that this can be done: For each variable $x \in \mathcal{V}$ there is $y \in \mathcal{V}$ such that $x < y$ (this is an assumption we made in Section 2)

2. $f = z(k_1, \ldots, k_m)$,
z does not occur in $k_1, \ldots, k_m$, and
there is a context Γ such that for all $k_i \in \mathcal{P}$, $\Gamma \vdash k_i{:}t_i^{a_i}$ (mod α) and FV$(f) \subseteq$ dom(Γ),
or

3. *$f = \neg f'$ and f' is a matrix (up to α-equality), or*
$f = f_1 \vee f_2$, there are contexts Γ_i and types t_i such that $\Gamma_i \vdash f_i : t_i (\text{mod } \alpha)$, and $\Gamma_1 \cup \Gamma_2$ is a context.

PROOF:

$\Rightarrow$ If f is simple, then use Theorem 7.16.
If $f = \neg f_1$ or $f = f_1 \vee f_2$, f a matrix in the context Γ (up to α-equality), then we have to prove that f_1 and f_2 are matrices upto α-equality. We'll give the proof for f_1.
Let $g_1, \ldots, g_m$ be the simple submatrices of f_1.
If $g_j = R(i_1, \ldots, i_{a(R)})$, then use Theorem 7.16(*) to see that g_j is a matrix in the context Γ (up to α-equality).
If $g_j = z(k_1, \ldots, k_n)$, then use Theorem 7.16(**) and Lemma 8.8 to see that g_j is a matrix in the context Γ (up to α-equality).
With the help of Lemma 8.10 (for the introduction of the $\vee$-symbols) and with rule 2 of RTT (for the introduction of the $\neg$-symbols) we see that f_1 is a matrix in the context Γ (up to α-equality).

$\Leftarrow$ 1. This is immediately clear for pseudomatrices of the form $R(i_1, \ldots, i_{a(R)})$.

2. Determine u^b such that $z : u^b \in \Gamma$.
Let $\Gamma' = \Gamma \setminus \{z{:}u^b\}$.
Define $u_i^{b_i}$ for all i as follows:

- If $k_i \in \mathcal{A}$ then $u_i^{b_i} \stackrel{\text{def}}{=} 0^0$;
- if $k_i \in \mathcal{V}$ then $u_i^{b_i}$ is the unique t^a such that $k_i{:}t^a \in \Gamma'$ (as $k_i \neq z$, $k_i \in$ dom(Γ));
- if $k_i \in \mathcal{P}$ then $u_i^{b_i}$ is the unique t^a such that $\Gamma' \vdash k_i{:}t^a$ (as z does not occur in k_i, k_i is a matrix in Γ')

Let $\Gamma'' \stackrel{\text{def}}{=} \Gamma' \cup \{z : (u_1^{b_1}, \ldots, u_m^{b_m})^{\max(b_1, \ldots, b_m)+1}{}_i\}$.
With Lemma 8.8: f is a matrix in the context Γ'' (up to $\alpha_{\Gamma''}$-equality).

3. Assume $f = \neg f'$ and f' is a matrix. With the use of rule 2 of RTT, it follows immediately that also f is a matrix.
Assume $f = f_1 \vee f_2$, f_1 and f_2 are matrices in contexts Γ_1 and Γ_2, and $\Gamma = \Gamma_1 \cup \Gamma_2$ is a context. By the method presented in the proof of Lemma 8.10, it is possible to make a derivation of f in the context Γ (up to α-equality).

□

We can now answer the question whether our matrices (as defined in 3.10) are the same as the matrices of the Principia.
First of all, we must notice that all the matrices from Definition 3.10 are also matrices of the Principia: This was motivated directly after we defined system RTT.
Moreover, we proved (in 8.12) that if f is a pseudomatrix, then the only reasons why f cannot be a matrix (according to Definition 3.10) are:

- There is a submatrix[25] $z(k_1, \ldots, k_m)$ of f in which z occurs in one of the k_i's;
- There is a submatrix $z(k_1, \ldots, k_m)$ of f and a $j \in \{1, \ldots, m\}$ such that k_j is a pseudomatrix, but not a matrix.

[25] The definition of submatrix can be found in 2.6

Pseudomatrices of the first type cannot be matrices in the Principia, because of the vicious circle principle.
Pseudomatrices f of the second type cannot be matrices in the Principia, because the only pseudomatrices that can occur in $z(k_1, \ldots, k_m)$ must be matrices in the Principia.

We conclude that we have described the matrices of the Principia Mathematica with the formal system RTT.

9 The λ-notation revisited

In Section 5 we introduced a different notation for pseudomatrices, based on λ-terms. In this section we show that this λ-notation has many advantages to our original notation, not only with respect to reduction, but also with respect to contexts and free variables.

First we extend the definition of "corresponding λ-term" of Section 5 to substitution trees:

Definition 9.1 If f is a pseudomatrix, then we will write $\tilde{f}$ for the corresponding λ-term (that was defined in Section 5).
If S is a substitution, then we will write $\tilde{S}$ for the corresponding λ-term (also defined in Section 5).
We now extend the notion of $\tilde{S}$ to substitution trees:
If T_1 and T_2 are substitution trees with corresponding λ-terms $\tilde{T_1}$ and $\tilde{T_2}$, then $T_1 \widetilde{\vee} T_2 \stackrel{\text{def}}{=} \vee \tilde{T_1} \tilde{T_2}$.
If T is a substitution tree with corresponding λ-term $\tilde{T}$, then $\widetilde{\neg T} \stackrel{\text{def}}{=} \neg \tilde{T}$.

Remark 9.2 The reader might wonder why $\tilde{f}$ is defined as above. At first sight it looks more reasonable to define it in another way, for instance, for $f = \mathtt{R}(\mathtt{x_1}, \mathtt{x_2})$ one might want to define $\tilde{f} = \lambda \mathtt{x_1 x_2}.\mathtt{R x_1 x_2}$.
The reason that we do not define it that way has immediately to do with substitution. If we want (in this example) to substitute $\mathtt{1}$ for $\mathtt{x_2}$ in order to obtain $\mathtt{R}(\mathtt{x_1}, \mathtt{1})$, the notation $\lambda \mathtt{x_1 x_2}.\mathtt{R x_1 x_2}$ and the notion of β-reduction forces us to substitute something for $\mathtt{x_1}$ as well, and we might not want to substitute something for $\mathtt{x_1}$ (except, maybe, $\mathtt{x_1}$ itself).
Another motivation for our definition can be found in the following lemma.

Lemma 9.3 *If f is a pseudomatrix then the free variables of f are exactly the free variables of the λ-term $\tilde{f}$.*

PROOF: Trivial. □

Lemma 9.4 *If f and g are pseudomatrices, and $\tilde{f} \equiv \tilde{g}$, then $f = g$.*

PROOF: Trivial. □

The λ-notation not only gives an extra motivation for the definition of free variables of a pseudomatrix (this definition was already discussed in Remark 2.4) but also provides a better look on the double rôle that variables can play in pseudomatrices.

Example 9.5 *Look at the pseudomatrix $f \stackrel{\text{def}}{=} \mathtt{z}(\mathtt{x}, \mathtt{R}(\mathtt{x}))$. It seems as if there are only two different variables in this pseudomatrix: $\mathtt{z}$ and $\mathtt{x}$.*
But when we look at $\tilde{f} = \mathtt{z x}(\lambda \mathtt{x}.\mathtt{R x})$, we see that $\mathtt{x}$ plays a double rôle: Its first occurrence is free, while its last occurrence is bound by the λ. It would have been better if we had taken two different variables for the two occurrences of $\mathtt{x}$, and to write, for instance, $\mathtt{z}(\mathtt{x_1}, \mathtt{R}(\mathtt{x_2}))$ and $\mathtt{z x_1}(\lambda \mathtt{x_2}.\mathtt{R x_2})$ instead.

The convention to take different variables for bound and free variables is well known in λ-calculus as the "variable-convention" or "Barendregt-convention" (see for instance [1]). We can assume this convention also for our pseudomatrix in the following form:

"If we have a pseudomatrix of the form $z(k_1, \ldots, k_m)$, and $k_i \in \mathcal{P}$ for a certain i, then the variables in k_i are supposed to be different from the free variables in $z(k_1, \ldots, k_m)$."

We use this convention also when handling substitutions:

"If $S = f\{x_i{:=}g_i\}$ is a substitution, then the variables occurring in the g_i's are supposed to be different from the free variables in f."

The Principia allow us to introduce a variable convention, as there is written[26]: *"we will write the function itself '$\varphi\hat{x}$' (Any other letter may be used in place of x)"* (The notation $\varphi\hat{x}$ is used in the Principia to indicate that φ is a propositional function (we would say: pseudomatrix) with one free variable: x.). It appears that in the Principia, no distinction is made between propositional functions (pseudomatrices) that are, in our terminology, α-equal (see Definition 2.5).

If we accept this variable convention, then our notion of reduction is almost the same as β-reduction in λ-calculus, as can be seen in the following lemmas. β-reduction will be denoted by $\rightarrow_\beta$ and $\twoheadrightarrow_\beta$ as usual.

Lemma 9.6 *Let S be a substitution tree, and $S \longrightarrow T$. Then $\tilde{S} \twoheadrightarrow_\beta \tilde{T}$.*

PROOF: From the definition of reduction, 4.6. We use the enumeration and the notations of 4.6.

1. S is a substitution $f\{x_i{:=}g_i\}$.
   (a) $f = R(i_1, \ldots, i_{a(R)})$. As $S \longrightarrow T$, all g_i's must be either variables or individuals, therefore
   $$\begin{aligned} \tilde{S} &= (\lambda x_1 \ldots x_n . R i_1 \ldots i_{a(R)}) g_1 \ldots g_n \\ &\twoheadrightarrow_\beta (R i_1 \ldots i_{a(R)})[x_1{:=}g_1] \ldots [x_n{:=}g_n] \end{aligned}$$
   Thanks to the variable convention, this is equivalent to $R i'_1 \ldots i'_{a(R)}$.
   And this is of course equal to $\tilde{T}$.
   (b) $f = z(k_1, \ldots, k_m)$. The "easy" case is handled in the same way as 1(a).
   For the hard case: Let $g'_i = g_i$ if $g_i \in \mathcal{A} \cup \mathcal{V}$; $g'_i = \lambda x_1 \ldots x_p . \widetilde{g_i}$ if $g_i \in \mathcal{P}$ and $x_1 < \ldots < x_p$ are the free variables of g_i.
   Note that $T = z'\{y_i{:=}k'_i\}$, hence
   $$\tilde{T} = (\lambda y_1 \ldots y_m . \widetilde{z'}) k''_1 \ldots k''_m,$$
   where $k''_j = \widetilde{k'_j}$ if k'_j is variable or individual; $k''_j = \lambda z_1 \ldots z_p . \widetilde{k'_j}$ if k'_j is pseudomatrix with free variables $z_1, \ldots, z_p$ (in alphabetical order).
   Due to the variable convention, also
   $$\begin{aligned} \tilde{S} &= (\lambda x_1 \ldots x_n . z k_1 \ldots k_m) g'_1 \ldots g'_n \\ &\twoheadrightarrow_\beta (\lambda y_1 \ldots y_m . \widetilde{z'}) k''_1 \ldots k''_m. \end{aligned}$$
   (c) Easy.
2. S is a substitution tree. Easy.

□

Lemma 9.7 *Let Γ be a context, and S a well-typed substitution tree in this context. Assume $\tilde{S} \twoheadrightarrow_\beta M$ for some λ-term M. Then there is a substitution tree T such that $S \longrightarrow_\Gamma T$ and $M \twoheadrightarrow_\beta \tilde{T}$.*

PROOF: Note that S reduces to a normal form, according to Theorem 7.17, say T is this normal form.
By a repeated application of Lemma 9.6, $\tilde{S} \twoheadrightarrow_\beta \tilde{T}$.
Now use the Church-Rosser theorem for β-reduction and the fact that $\tilde{T}$ is in β-normal form to obtain: $M \twoheadrightarrow_\beta \tilde{T}$. □

[26] Introduction to the 2nd edition, Chapter II, Section II, page 40.

When we look at typed λ-calculus, we see that much information on types is "stored" behind the λs.
In our system RTT, the contexts sometimes play a clumsy rôle. Especially in the rules 3, 4 and 6, where the context Γ' of the conclusion is defined in an indirect way. The main reason for this is that contexts also contain type-information on bound (non-free) variables.
If we store the information on the non-free variables "behind the λ's", contexts only need to contain information on free variables. Then, in rules 3 and 4, Γ' can be replaced by $\Gamma \cup \{z : u\}$; in rule 6, Γ' can be replaced by Γ (see also Theorem 7.6).
Moreover, in rule 3 and 6 the contexts of minor and major premise can *differ* now (and will differ, according to the variable convention).

Rules 3, 4 and 6 now become:

3. If $\Gamma \vdash f : (t_1^{a_1}, \ldots, t_m^{a_m})^a$, and $k \in \mathcal{A} \cup \mathcal{P}$ is an argument of f, and $\Delta \vdash k : t_{m+1}^{a_{m+1}}$, and $y < z$ for all $y \in \mathrm{dom}(\Gamma)$, then

$$\Gamma \cup \{z : t_{m+1}^{a_{m+1}}\} \vdash h : (t_1^{a_1}, \ldots, t_{m+1}^{a_{m+1}})^{\max(a, a_{m+1}+1)}$$

   Here, h is the pseudomatrix obtained by replacing the arguments k' of f which are α_Γ-equal to k by $zx_1 \ldots x_p$ (where $x_1 < \ldots < x_p$ are the free variables of the k' in question).

4. If $\Gamma \vdash f : (t_1^{a_1}, \ldots, t_m^{a_m})^a$, and $\Gamma \vdash g : t_{m+1}^{a_{m+1}}$, and $y < z$ for all $y \in \mathrm{dom}(\Gamma)$, then

$$\Gamma \cup \{z : t_{m+1}^{a_{m+1}}\} \vdash h : (t_1^{a_1}, \ldots, t_{m+1}^{a_{m+1}})^{\max(a, a_{m+1}+1)}.$$

   Here, h is the pseudomatrix obtained by replacing the submatrices g' of f which are α_Γ-equal to g by $zx_1 \ldots x_p$ (where $x_1 < \ldots < x_p$ are the free variables of the g' in question).

6. If y is the ith free variable in f (according to the alphabetical order), and $\Gamma \cup \{y : t_i^{a_i}\} \vdash f : (t_1^{a_1}, \ldots, t_n^{a_n})^a$, and $\Delta \vdash k : t_i^{a_i}$, then

$$\Gamma \vdash f[y{:=}k]_{\Gamma \cup \{y : t_i^{a_i}\}} : (t_1^{a_1}, \ldots, t_{i-1}^{a_{i-1}}, t_{i+1}^{a_{i+1}}, \ldots, t_n^{a_n})^b$$

   Here, $b = 1 + \max(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n)$.

Remark 9.8 In the new substitution-rule 6, there is no relation between the contexts Γ and Δ any more, and in the conclusion of this rule, Δ is not present. This can be done because in the substitution, all the free variables of k become bound, and therefore, type-information from Δ of such a variable is stored behind the λ that binds it. The information in Δ has become superfluous for the conclusion.

Example 9.9 *In typed λ-calculus we can write* $\mathtt{zx}(\lambda \mathtt{x}{:}\mathtt{0}^{\mathtt{0}}.\mathtt{Rx})$ *instead of* $\mathtt{z}(\mathtt{x}, \mathtt{R}(\mathtt{x}))$ *(in a context where* x *has type* $\mathtt{0}^{\mathtt{0}}$, z *has type* $(\mathtt{0}^{\mathtt{0}}, (\mathtt{0}^{\mathtt{0}})^{\mathtt{1}})^{\mathtt{2}}$*).*
We can derive this pseudomatrix as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \mathtt{R}(\mathtt{0}) : ()^{\mathtt{0}}}3}{\mathtt{x}{:}\mathtt{0}^{\mathtt{0}} \vdash \mathtt{R}(\mathtt{x}) : (\mathtt{0}^{\mathtt{0}})^{\mathtt{1}}}4}{\mathtt{x}{:}\mathtt{0}^{\mathtt{0}}, \mathtt{y}{:}(\mathtt{0}^{\mathtt{0}})^{\mathtt{1}} \vdash \mathtt{yx} : (\mathtt{0}^{\mathtt{0}}, (\mathtt{0}^{\mathtt{0}})^{\mathtt{1}})^{\mathtt{2}}}4}{\mathtt{x}{:}\mathtt{0}^{\mathtt{0}}, \mathtt{y}{:}(\mathtt{0}^{\mathtt{0}})^{\mathtt{1}}, \mathtt{z}{:}(\mathtt{0}^{\mathtt{0}}, (\mathtt{0}^{\mathtt{0}})^{\mathtt{1}})^{\mathtt{2}} \vdash \mathtt{zxy} : (\mathtt{0}^{\mathtt{0}}, (\mathtt{0}^{\mathtt{0}})^{\mathtt{1}}, (\mathtt{0}^{\mathtt{0}}, (\mathtt{0}^{\mathtt{0}})^{\mathtt{1}})^{\mathtt{2}})^{\mathtt{3}}}4 \qquad \dfrac{\dfrac{}{\vdash \mathtt{R}(\mathtt{0}) : ()^{\mathtt{0}}}3}{\mathtt{x}'{:}(\mathtt{0}^{\mathtt{0}}) \vdash \mathtt{R}(\mathtt{x}') : (\mathtt{0}^{\mathtt{0}})^{\mathtt{1}}}3}{\mathtt{x}{:}(\mathtt{0}^{\mathtt{0}}), \mathtt{z}{:}(\mathtt{0}^{\mathtt{0}}, (\mathtt{0}^{\mathtt{0}})^{\mathtt{1}})^{\mathtt{2}} \vdash \mathtt{zx}(\lambda \mathtt{x}'{:}\mathtt{0}^{\mathtt{0}}.\mathtt{Rx}') : (\mathtt{0}^{\mathtt{0}}, (\mathtt{0}^{\mathtt{0}}, (\mathtt{0}^{\mathtt{0}})^{\mathtt{1}})^{\mathtt{2}})^{\mathtt{3}}}6$$

The following variants of Lemma 3.16 are easy to prove. The first lemma can be seen as an incarnation of the "free variable lemma" in typed λ-calculus; the second lemma as a variant of the "thinning lemma".

Lemma 9.10 *If* $\Gamma \vdash f : t^a$ *then* $\mathrm{FV}(f) \subseteq \mathrm{dom}(\Gamma)$.

PROOF: Induction on $\Gamma \vdash f : t^a$. □

Lemma 9.11 *If* $\Gamma \vdash t^a$ *then there is a context* Δ *such that* $\mathrm{dom}(\Delta) = \mathrm{FV}(f)$ *and* $\Delta \vdash f : t^a$.

PROOF: Induction on $\Gamma \vdash f : t^a$. □

When we look at our renewed system, we see that it has many characteristics of the type systems in, for example, the Barendregt cube (see [2]): Rules 1 and 2 provide a "start", rules 3 and 4 together can be seen as an abstraction rule, rule 5 is a weakening rule and rule 6 can be seen as an application rule.
We have variants of well-known theorems as

- Church-Rosser (4.8)
- Strong normalisation (6.3, 7.17)
- Unicity of types (7.9)
- Free variable lemma (9.10)
- Thinning lemma (9.11)

There are of course also differences:

- In system RTT, the notion of reduction is hidden in the system: the system itself gives only pseudomatrices in normal form. This is not the case in cube-like systems. We also use a very restricted part of λ-calculus.
- Our notion of type is a very restricted one, even if we compare it to the types of the simplest system in the Barendregt cube, $\lambda \rightarrow$.
- Another difference with modern type systems is that we use constants of various kinds ($\mathtt{0}, \mathtt{1}, \ldots$, but also $\mathtt{R}, \mathtt{S}, \vee, \neg$) in an explicit way.
- Variables are not pseudomatrices. In modern type systems, a variable is a term. We cannot have this in our system. In [10], Russell writes[27]: "*A variable is not* any term *simply, but any term as entering into a propositional function.*"
  In system RTT, we can derive, for instance,

  $$\mathtt{z} : ()^0 \vdash \mathtt{z}() : (()^0)^1$$

  while, regarding modern type systems, one should expect

  $$\mathtt{z} : ()^0 \vdash \mathtt{z} : ()^0.$$

10 Formulas

We have not introduced quantifiers yet, though quantifiers are important symbols in logic.
Quantifiers are the last step towards the definition of *(propositional) functions*, which are the final objects in our Principia-inspired system.

Definition 10.1 (Formulas)

- Each matrix is a formula;
- If f is a matrix of type $(t_1^{a_1}, \ldots, t_n^{a_n})^a$ in the context Γ, Q is one of the quantifiers $\exists, \forall$, y is the ith free variable of f (according to the alphabetic order), then

  $$Qy{:}t_i^{a_i}[f]$$

  is a formula of type $(t_1^{a_1}, \ldots, t_{i-1}^{a_{i-1}}, t_{i+1}^{a_{i+1}}, \ldots, t_n^{a_n})^a$ in the context $\Gamma \setminus \{y : t_i^{a_i}\}$. Of course, $\mathrm{FV}(Qy{:}t_i^{a_i}[f]) = \mathrm{FV}(f) \setminus \{y\}$.
  Like we are used to do with matrices, we write this as a judgement:

  $$\Gamma \setminus \{y{:}t_i^{a_i}\} \vdash Qy{:}t_i^{a_i}[f] : (t_1^{a_1}, \ldots, t_{i-1}^{a_{i-1}}, t_{i+1}^{a_{i+1}}, \ldots, t_n^{a_n})^a.$$

[27] Chapter VIII: "The variable", p.94 of the 7th impression.

We can extend theorem 7.6 in an obvious way:

Theorem 10.2 *Assume* Γ *is a context,* f *is a formula with free variables* $x_1 < \ldots < x_m$ *and* $\Gamma \vdash f : (t_1^{a_1}, \ldots, t_n^{a_n})^a$. *Then* $m = n$ *and* $x_i : t_i^{a_i} \in \Gamma$.

PROOF: By theorem 7.6 and the definition of formula. □

Remark 10.3 Note that lemma 7.8 does *not* hold for formulas in general! $a_1, \ldots, a_n$ (notation of 7.8) are the orders of the free variables in f. It is possible that a formula f obtains a quantifier over a variable of an order higher than the a_is. Then a will also be higher than $\max(a_1, \ldots, a_n) + 1$.

The definition of substitution can be extended to formulas, too. We will not work this out in detail.

Remark 10.4 A strange consequence of this system is, that things like $\forall \mathbf{x} : 0^0[\mathrm{R}(\mathbf{x})] \vee \exists \mathbf{x} : 0^0[\neg \mathrm{R}(\mathbf{x})]$ are not a formula. This is noticed in the second edition of [16][28], where, for instance, $\forall \mathbf{x} : 0^0[\mathrm{R}(\mathbf{x})] \vee \exists \mathbf{x} : 0^0[\neg \mathrm{R}(\mathbf{x})]$ is *defined* to be $\forall \mathbf{x} : 0^0 \exists \mathbf{y} : 0^0[\mathrm{R}(\mathbf{x}) \vee \neg \mathrm{R}(\mathbf{y})]$.
Our system only gives formulas that are in prenex normal form, and other formulas are defined by their equivalent prenex normal form.
There is a problem with these kind of definitions: The system is now assuming certain *logical* rules or axioms (in this example: the formulas $\forall \mathbf{x}{:}0^0[\mathrm{R}(\mathbf{x})] \vee \exists \mathbf{x}{:}0^0[\neg \mathrm{R}(\mathbf{x})]$ and $\forall \mathbf{x}{:}0^0 \exists \mathbf{y}{:}0^0[\mathrm{R}(\mathbf{x}) \vee \neg \mathrm{R}(\mathbf{y})]$ are equivalent). Therefore it might give some undesired results when a logical system is used that does not agree with these rules or axioms. In this case, for instance, problems will arise when we want to use intuitionistic logic, where the equivalence of the formulas $\forall \mathbf{x} : 0^0[\mathrm{R}(\mathbf{x})] \vee \exists \mathbf{x} : 0^0[\neg \mathrm{R}(\mathbf{x})]$ and $\forall \mathbf{x} : 0^0 \exists \mathbf{y} : 0^0[\mathrm{R}(\mathbf{x}) \vee \neg \mathrm{R}(\mathbf{y})]$ does not hold in general, see for instance [14].
There is an easy solution to this problem: Just skip the notion of matrix, substitute the word "formula" for the word "matrix" everywhere in the definition RTT and add an extra rule:

8. If $\Gamma \cup \{y : t_i^{a_i}\} \vdash f : (t_1^{a_1}, \ldots, t_m^{a_m})^n$ and y is the ith free variable in f, then, for any $Q \in \{\forall, \exists\}$,

$$\Gamma \vdash Qy{:}t_i^{a_i}[f] : (t_1^{a_1}, \ldots, t_{i-1}^{a_{i-1}}, t_{i+1}^{a_{i+1}}, \ldots, t_m^{a_m})^n.$$

Rule 6 must be modified now, as the order of a formula also depends on the order of the variables which are bound by a quantifier. The order of the resulting formula f should be $1 + a$, where a is the maximum of the orders of all free variables and all variables bound by a quantifier in f.
With this adaption, also an expression like $\forall \mathbf{x}{:}0^0[\mathrm{R}(\mathbf{x})] \vee \exists \mathbf{x}{:}0^0[\neg \mathrm{R}(\mathbf{x})]$ becomes a formula, and does not need to have the same interpretation as the expression $\forall \mathbf{x}{:}0^0 \exists \mathbf{y}{:}0^0[\mathrm{R}(\mathbf{x}) \vee \neg \mathrm{R}(\mathbf{y})]$.

We must realize that the introduction of this rule 8 yields also some other objects. Up till now, all the variables that occur in formulas have a free type. With rule 8 it is possible to introduce also variables of a non-free type (use rule 4 of RTT). This is a serious extension of RTT, which is certainly not present in the Principia, in which only variables for matrices are allowed.
On the other hand, we certainly would like to have expressions like $z(\forall x{:}t^a[g])$, where g is a pseudomatrix such that $\forall x{:}t^a[g]$ has a free type. The way to obtain such expressions would be substitution of $\forall x{:}t^a[g]$ for y in a pseudomatrix $z(y)$.
We can solve both problems by making restrictions to the use of the rules 3, 4 and 6 of RTT. If we demand that the type of the minor premise in these rules must be a *free* type, no variables of a non-free type can be introduced, while substitutions in the style described above are still possible.

Another reason why Russell and Whitehead presented their system not in this way might be that they wanted to stress the fact that the order of a formula depends on the orders of both the free variables and the variables bound by quantifiers that occur in it. To compute the order of a formula (in prenex normal form), one should compute the order of this formula *without* quantifiers: Adding the quantifiers afterwards does not change the order of the formula.

[28] Introduction to the Second Edition, section III: General propositions of limited scope, pp. xxiv-xxv, and Appendix A, giving a chapter *8 which should replace the "old" chapter *9.

11 The Axiom of Reducibility

We can define a notion of truth in Tarski's style (see [12]) on our formal system. Then our system is complete and we can play with it.
It appears that the system isn't easy to handle with. The main reason for this is the use of the ramification. We illustrate this with an example.

Example 11.1 (Equality) *One tends to define the notion of* equality *in the style of Leibniz* [7]*:*

$$\mathtt{x} = \mathtt{y} \overset{\text{def}}{\leftrightarrow} \forall \mathtt{z}[\mathtt{z}(\mathtt{x}) \leftrightarrow \mathtt{z}(\mathtt{y})],$$

or in words: Two individuals are equal if and only if they have exactly the same properties. Unfortunately, the formula $\forall \mathtt{z}[\mathtt{z}(\mathtt{x}) \leftrightarrow \mathtt{z}(\mathtt{y})]$ *is not a formula of our formal system, whatever context one chooses. We have to write* $\forall \mathtt{z} : (\mathtt{o}^{\mathtt{o}})^n[\mathtt{z}(\mathtt{x}) \leftrightarrow \mathtt{z}(\mathtt{y})]$ *for some* $n > 1$*, i.e., we have to specify the order of* $\mathtt{z}$ *beforehand. And when two things have the same order-n-properties, it is not sure that they will also have the same order-*$(n+1)$*-properties.*

Russell and Whitehead tried to solve the problems of the previous example with the so-called *axiom of reducibility*. In the following, we translate the relevant notions of the Principia in our setting.

Definition 11.2 Assume, Γ is a context, f is a formula and $\Gamma \vdash f : t^a$ for some type t^a. Assume $\mathrm{FV}(f) = \{x_1, \ldots, x_n\}$ and $\{x_1 : t_1^{a_1}, \ldots, x_n : t_n^{a_n}\} \subseteq \Gamma$. f is called *predicative* if $a = \max(a_1, \ldots, a_n)$. Otherwise, f is called *impredicative*.[29]

Axiom 11.3 (Axiom of Reducibility) *For each formula* f*, there is a* predicative *formula* g *such that* f *and* g *are equivalent.*

Accepting this axiom, it is easy to define equality on individuals: One just uses the formula

$$\forall \mathtt{z} : (\mathtt{o}^{\mathtt{o}})^1[\mathtt{z}(\mathtt{x}) \leftrightarrow \mathtt{z}(\mathtt{y})].$$

If f is a function of type $(\mathtt{o}^{\mathtt{o}})^n$ (for some $n > 0$), and $\mathtt{a}$ and $\mathtt{b}$ are individuals for which $\mathtt{a} = \mathtt{b}$ holds (according to this last definition) then also $f(\mathtt{a}) \leftrightarrow f(\mathtt{b})$ holds, even if $n \neq 1$: With the Axiom of Reducibility we can determine a predicative function g (so: of type $(\mathtt{o}^{\mathtt{o}})^1$), equivalent to f. As g has order 1, $g(\mathtt{a}) \leftrightarrow g(\mathtt{b})$ holds. And because f and g are equivalent, also $f(\mathtt{a}) \leftrightarrow f(\mathtt{b})$ holds.
So with the axiom of reducibility it is possible to define equality in the style of Leibniz.

An axiom, however, has to be defended. The only argument in favour of the axiom of reducibility that is given in the "Principia" is, that "it works". In the introduction to the 2nd edition, the writers admit: "*This axiom has a purely pragmatic justification: it leads to the desired results, and to no others. But clearly it is not the sort of axiom with which we can rest content.*" This is not very satisfying, and therefore is the main objection against the axiom.
Weyl [15] states that "*if the properties are constructed there is no room for an axiom here; it is a question which ought to be decided on ground of the construction*", and that "*with his axiom of reducibility Russell therefore abandoned the road of logical analysis and turned from the constructive to the existential-axiomatic standpoint*". Since we *do* indeed construct formulas, we should take this objection rather serious.

Another solution to get rid of the ramification was presented by Ramsey [9] and Hilbert and Ackermann [8]. The ramified type theory was invented to keep away the paradoxes that appeared in Frege's "Begriffschrift".
Ramsey and Hilbert and Ackermann divide these paradoxes in two parts:

- The contradictions that are removed by pointing out that (because of the Vicious Circle Principle 1.1) a propositional function cannot significantly take itself as argument, such as the antinomies of Russell, Burali-Forti and Cantor. These paradoxes can be prevented without the use of orders in type theory.

[29] cf. [16], Introduction, Chapter II, Section v, p. 53.

- The contradictions resulting from ambiguities of language (Richard, Epimenides). Hilbert and Ackermann call these paradoxes *semantical*. These contradictions are not pure logical contradictions, but also based on natural language, which they call a *meta-language*, a language which speaks about the formal language of mathematics. Of course, it is also possible to talk about this metalanguage, in a meta-metalanguage, and so on. Propositions which are partly in the formal language, and partly in meta-language are therefore meaningless, and this solves the second class of paradoxes.

So with introducing the notion of metalanguage, the use of types with orders (ramified types) is not necessary to avoid the paradoxes. One can leave out the orders and thus obtain the so-called *simple* type-theory. Simple type theory then becomes the standard for future type systems. The basis for these future systems was laid in Church's formulation of the simple theory of types [4].
Of course, it is easy to base a formal system of the simple type theory on our system of ramified type theorie: One just has to remove all the "superscripts" in the formulas, deduction-rules of RTT, types and contexts.

12 Conclusions

In the first sections, we presented a formal type system that remains close to the Principia Mathematica (RTT, 3.9). In order to make this system we had to take a closer look on substitution in general (Section 4). We formalized the intuitive notion of substitution that is present in the Principia, and showed its proper behaviour in general (Theorem 6.3), with respect to free variables (Theorem 6.7) and with respect to types (Theorem 7.17, Corollary 7.18).
We showed the relation between this substitution and β-reduction in λ-calculus (Section 5 and Section 9), and made a comparison between RTT and modern type systems in λ-calculus.
In Section 8 we compared our notion of matrix to the one in the Principia, and at the end of this section, we showed that these notions are exactly the same.
We extended our system to formulas by adding quantifiers (Section 10). Thus we obtained a formal system that exactly represents the "propositional functions" and the "propositions" of the Principia.
The ramified theory of types is a very restrictive system for the development of mathematics, as only predicative formluas are allowed and also impredicative formluas are used throughout mathematics. A first solution to this problem was the Axiom of Reducibility. In Section 11 we argue that, with respect to RTT, this axiom is unacceptable. We also describe the second solution, the simple theory of types, and describe how RTT can be appended to form a formal system for this simple type theory.

We only discussed the type theory of the Principia Mathematica in this paper, and didn't look at other, more modern type systems. We gave only a quick glance over the simple type theory as proposed by Ramsey, Hilbert and Ackermann, and didn't discuss Church's formalization of the simple type theory.
Another point for further research can be the embedding of the above system in one of the systems of the Barendregt cube. At that stage, however, it will be hardly possible to maintain the exact rules 1–7 of our system RTT and, with them, the philosophy and motivation that is behind the original system of Russell and Whitehead.

Acknowledgements

I would like to thank the following people for their help, their suggestions and their patience: Roel Bloo, Herman Geuvers, Kees Hemerik (who gave me the idea to make the above formalization), Fairouz Kamareddine, Rob Nederpelt, Eric Poll (for suggesting the lambda notation of Section 5).

References

[1] H.P. Barendregt. *Lambda Calculus: its Syntax and Semantics.* North-Holland, Amsterdam, revised edition, 1984.

[2] H.P. Barendregt. Lambda calculi with types. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of logic in Computer Science, vol. 2*, pages 117–309. Oxford University Press, 1992.

[3] E.W. Beth. *The Foundations of Mathematics.* Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam, 1959.

[4] A. Church. A formulation of the simple theory of types. *The Journal of Symbolic Logic*, 5:56–68, 1940.

[5] A.A. Fraenkel, Y. Bar-Hillel, and A. Levy. *Foundations of Set Theory.* Studies in Logic and the Foundations of Mathematics **67**. North Nolland, Amsterdam, second edition, 1973.

[6] G. Frege. *Begriffschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens.* Nebert, Halle, 1879. German; also in [13].

[7] C.I. Gerhardt, editor. *Die Philosophischen Schriften von Gottfried Wilhelm Leibniz.* , Berlin, 1890.

[8] D. Hilbert and W. Ackermann. *Grundzüge der Theoretischen Logik.* Die Grundlehren der Mathematischen Wissenschaften in Einzeldarstellungen, Band XXVII. Springer Verlag, Berlin, first edition, 1928. (German).

[9] F.P. Ramsey. The foundations of mathematics. *Proceedings of the London Mathematical Society*, pages 338–384, 1926.

[10] B. Russell. *The Principles of Mathematics.* Allen & Unwin, London, 1903.

[11] B. Russell. Mathematical logic as based on the theory of types. *American Journal of Mathematics*, 30, 1908. Also in [13].

[12] A. Tarski. Der Wahrheitsbegriff in den formalisierten Sprachen. *Studia Philosophica*, 1:261–405, 1936. German translation by L. Blauwstein from the Polish original (1933) with a postscript added.

[13] J. van Heijenoort, editor. *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931.* Harvard University Press, Cambridge, Massachusetts, 1967.

[14] W.H.M. Veldman. Intuïtionistische wiskunde. Lecture Notes, Catholic University Nijmegen, 1991. (Dutch).

[15] H. Weyl. Mathematics and logic: A brief survey serving as preface to a review of "the philosophy of Bertrand Russell". *American Mathematical Monthly*, 1946.

[16] A.N. Whitehad and B. Russell. *Principia Mathematica.* Cambridge University Press, 1910. (All references are to the first volume, unless otherwise indicated).

Computing Science Reports

Department of Mathematics and Computing Science
Eindhoven University of Technology

In this series appeared:

91/01	D. Alstein	Dynamic Reconfiguration in Distributed Hard Real-Time Systems, p. 14.
91/02	R.P. Nederpelt H.C.M. de Swart	Implication. A survey of the different logical analyses "if...,then...", p. 26.
91/03	J.P. Katoen L.A.M. Schoenmakers	Parallel Programs for the Recognition of *P*-invariant Segments, p. 16.
91/04	E. v.d. Sluis A.F. v.d. Stappen	Performance Analysis of VLSI Programs, p. 31.
91/05	D. de Reus	An Implementation Model for GOOD, p. 18.
91/06	K.M. van Hee	SPECIFICATIEMETHODEN, een overzicht, p. 20.
91/07	E.Poll	CPO-models for second order lambda calculus with recursive types and subtyping, p. 49.
91/08	H. Schepers	Terminology and Paradigms for Fault Tolerance, p. 25.
91/09	W.M.P.v.d.Aalst	Interval Timed Petri Nets and their analysis, p.53.
91/10	R.C.Backhouse P.J. de Bruin P. Hoogendijk G. Malcolm E. Voermans J. v.d. Woude	POLYNOMIAL RELATORS, p. 52.
91/11	R.C. Backhouse P.J. de Bruin G.Malcolm E.Voermans J. van der Woude	Relational Catamorphism, p. 31.
91/12	E. van der Sluis	A parallel local search algorithm for the travelling salesman problem, p. 12.
91/13	F. Rietman	A note on Extensionality, p. 21.
91/14	P. Lemmens	The PDB Hypermedia Package. Why and how it was built, p. 63.
91/15	A.T.M. Aerts K.M. van Hee	Eldorado: Architecture of a Functional Database Management System, p. 19.
91/16	A.J.J.M. Marcelis	An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs, p. 25.

91/17	A.T.M. Aerts P.M.E. de Bra K.M. van Hee	Transforming Functional Database Schemes to Relational Representations, p. 21.
91/18	Rik van Geldrop	Transformational Query Solving, p. 35.
91/19	Erik Poll	Some categorical properties for a model for second order lambda calculus with subtyping, p. 21.
91/20	A.E. Eiben R.V. Schuwer	Knowledge Base Systems, a Formal Model, p. 21.
91/21	J. Coenen W.-P. de Roever J.Zwiers	Assertional Data Reification Proofs: Survey and Perspective, p. 18.
91/22	G. Wolf	Schedule Management: an Object Oriented Approach, p. 26.
91/23	K.M. van Hee L.J. Somers M. Voorhoeve	Z and high level Petri nets, p. 16.
91/24	A.T.M. Aerts D. de Reus	Formal semantics for BRM with examples, p. 25.
91/25	P. Zhou J. Hooman R. Kuiper	A compositional proof system for real-time systems based on explicit clock temporal logic: soundness and complete ness, p. 52.
91/26	P. de Bra G.J. Houben J. Paredaens	The GOOD based hypertext reference model, p. 12.
91/27	F. de Boer C. Palamidessi	Embedding as a tool for language comparison: On the CSP hierarchy, p. 17.
91/28	F. de Boer	A compositional proof system for dynamic proces creation, p. 24.
91/29	H. Ten Eikelder R. van Geldrop	Correctness of Acceptor Schemes for Regular Languages, p. 31.
91/30	J.C.M. Baeten F.W. Vaandrager	An Algebra for Process Creation, p. 29.
91/31	H. ten Eikelder	Some algorithms to decide the equivalence of recursive types, p. 26.
91/32	P. Struik	Techniques for designing efficient parallel programs, p. 14.
91/33	W. v.d. Aalst	The modelling and analysis of queueing systems with QNM-ExSpect, p. 23.
91/34	J. Coenen	Specifying fault tolerant programs in deontic logic, p. 15.

91/35	F.S. de Boer J.W. Klop C. Palamidessi	Asynchronous communication in process algebra, p. 20.
92/01	J. Coenen J. Zwiers W.-P. de Roever	A note on compositional refinement, p. 27.
92/02	J. Coenen J. Hooman	A compositional semantics for fault tolerant real-time systems, p. 18.
92/03	J.C.M. Baeten J.A. Bergstra	Real space process algebra, p. 42.
92/04	J.P.H.W.v.d.Eijnde	Program derivation in acyclic graphs and related problems, p. 90.
92/05	J.P.H.W.v.d.Eijnde	Conservative fixpoint functions on a graph, p. 25.
92/06	J.C.M. Baeten J.A. Bergstra	Discrete time process algebra, p.45.
92/07	R.P. Nederpelt	The fine-structure of lambda calculus, p. 110.
92/08	R.P. Nederpelt F. Kamareddine	On stepwise explicit substitution, p. 30.
92/09	R.C. Backhouse	Calculating the Warshall/Floyd path algorithm, p. 14.
92/10	P.M.P. Rambags	Composition and decomposition in a CPN model, p. 55.
92/11	R.C. Backhouse J.S.C.P.v.d.Woude	Demonic operators and monotype factors, p. 29.
92/12	F. Kamareddine	Set theory and nominalisation, Part I, p.26.
92/13	F. Kamareddine	Set theory and nominalisation, Part II, p.22.
92/14	J.C.M. Baeten	The total order assumption, p. 10.
92/15	F. Kamareddine	A system at the cross-roads of functional and logic programming, p.36.
92/16	R.R. Seljée	Integrity checking in deductive databases; an exposition, p.32.
92/17	W.M.P. van der Aalst	Interval timed coloured Petri nets and their analysis, p. 20.
92/18	R.Nederpelt F. Kamareddine	A unified approach to Type Theory through a refined lambda-calculus, p. 30.
92/19	J.C.M.Baeten J.A.Bergstra S.A.Smolka	Axiomatizing Probabilistic Processes: ACP with Generative Probabilities, p. 36.
92/20	F.Kamareddine	Are Types for Natural Language? P. 32.

92/21	F.Kamareddine	Non well-foundedness and type freeness can unify the interpretation of functional application, p. 16.
92/22	R. Nederpelt F.Kamareddine	A useful lambda notation, p. 17.
92/23	F.Kamareddine E.Klein	Nominalization, Predication and Type Containment, p. 40.
92/24	M.Codish D.Dams Eyal Yardeni	Bottum-up Abstract Interpretation of Logic Programs, p. 33.
92/25	E.Poll	A Programming Logic for Fω, p. 15.
92/26	T.H.W.Beelen W.J.J.Stut P.A.C.Verkoulen	A modelling method using MOVIE and SimCon/ExSpect, p. 15.
92/27	B. Watson G. Zwaan	A taxonomy of keyword pattern matching algorithms, p. 50.
93/01	R. van Geldrop	Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.
93/02	T. Verhoeff	A continuous version of the Prisoner's Dilemma, p. 17
93/03	T. Verhoeff	Quicksort for linked lists, p. 8.
93/04	E.H.L. Aarts J.H.M. Korst P.J. Zwietering	Deterministic and randomized local search, p. 78.
93/05	J.C.M. Baeten C. Verhoef	A congruence theorem for structured operational semantics with predicates, p. 18.
93/06	J.P. Veltkamp	On the unavoidability of metastable behaviour, p. 29
93/07	P.D. Moerland	Exercises in Multiprogramming, p. 97
93/08	J. Verhoosel	A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.
93/09	K.M. van Hee	Systems Engineering: a Formal Approach Part I: System Concepts, p. 72.
93/10	K.M. van Hee	Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.
93/11	K.M. van Hee	Systems Engineering: a Formal Approach Part III: Modeling Methods, p. 101.
93/12	K.M. van Hee	Systems Engineering: a Formal Approach Part IV: Analysis Methods, p. 63.
93/13	K.M. van Hee	Systems Engineering: a Formal Approach

		Part V: Specification Language, p. 89.
93/14	J.C.M. Baeten J.A. Bergstra	On Sequential Composition, Action Prefixes and Process Prefix, p. 21.
93/15	J.C.M. Baeten J.A. Bergstra R.N. Bol	A Real-Time Process Logic, p. 31.
93/16	H. Schepers J. Hooman	A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems, p. 27
93/17	D. Alstein P. van der Stok	Hard Real-Time Reliable Multicast in the DEDOS system, p. 19.
93/18	C. Verhoef	A congruence theorem for structured operational semantics with predicates and negative premises, p. 22.
93/19	G-J. Houben	The Design of an Online Help Facility for ExSpect, p.21.
93/20	F.S. de Boer	A Process Algebra of Concurrent Constraint Programming, p. 15.
93/21	M. Codish D. Dams G. Filé M. Bruynooghe	Freeness Analysis for Logic Programs - And Correctness?, p. 24.
93/22	E. Poll	A Typechecker for Bijective Pure Type Systems, p. 28.
93/23	E. de Kogel	Relational Algebra and Equational Proofs, p. 23.
93/24	E. Poll and Paula Severi	Pure Type Systems with Definitions, p. 38.
93/25	H. Schepers and R. Gerth	A Compositional Proof Theory for Fault Tolerant Real-Time Distributed Systems, p. 31.
93/26	W.M.P. van der Aalst	Multi-dimensional Petri nets, p. 25.
93/27	T. Kloks and D. Kratsch	Finding all minimal separators of a graph, p. 11.
93/28	F. Kamareddine and R. Nederpelt	A Semantics for a fine λ-calculus with de Bruijn indices, p. 49.
93/29	R. Post and P. De Bra	GOLD, a Graph Oriented Language for Databases, p. 42.
93/30	J. Deogun T. Kloks D. Kratsch H. Müller	On Vertex Ranking for Permutation and Other Graphs, p. 11.
93/31	W. Körver	Derivation of delay insensitive and speed independent CMOS circuits, using directed commands and production rule sets, p. 40.
93/32	H. ten Eikelder and H. van Geldrop	On the Correctness of some Algorithms to generate Finite Automata for Regular Expressions, p. 17.

93/33	L. Loyens and J. Moonen	ILIAS, a sequential language for parallel matrix computations, p. 20.
93/34	J.C.M. Baeten and J.A. Bergstra	Real Time Process Algebra with Infinitesimals, p.39.
93/35	W. Ferrer and P. Severi	Abstract Reduction and Topology, p. 28.
93/36	J.C.M. Baeten and J.A. Bergstra	Non Interleaving Process Algebra, p. 17.
93/37	J. Brunekreef J-P. Katoen R. Koymans S. Mauw	Design and Analysis of Dynamic Leader Election Protocols in Broadcast Networks, p. 73.
93/38	C. Verhoef	A general conservative extension theorem in process algebra, p. 17.
93/39	W.P.M. Nuijten E.H.L. Aarts D.A.A. van Erp Taalman Kip K.M. van Hee	Job Shop Scheduling by Constraint Satisfaction, p. 22.
93/40	P.D.V. van der Stok M.M.M.P.J. Claessen D. Alstein	A Hierarchical Membership Protocol for Synchronous Distributed Systems, p. 43.
93/41	A. Bijlsma	Temporal operators viewed as predicate transformers, p. 11.
93/42	P.M.P. Rambags	Automatic Verification of Regular Protocols in P/T Nets, p. 23.
93/43	B.W. Watson	A taxomomy of finite automata construction algorithms, p. 87.
93/44	B.W. Watson	A taxonomy of finite automata minimization algorithms, p. 23.
93/45	E.J. Luit J.M.M. Martin	A precise clock synchronization protocol,p.
93/46	T. Kloks D. Kratsch J. Spinrad	Treewidth and Patwidth of Cocomparability graphs of Bounded Dimension, p. 14.
93/47	W. v.d. Aalst P. De Bra G.J. Houben Y. Kornatzky	Browsing Semantics in the "Tower" Model, p. 19.
93/48	R. Gerth	Verifying Sequentially Consistent Memory using Interface Refinement, p. 20.

94/01	P. America M. van der Kammen R.P. Nederpelt O.S. van Roosmalen H.C.M. de Swart	The object-oriented paradigm, p. 28.
94/02	F. Kamareddine R.P. Nederpelt	Canonical typing and Π-conversion, p. 51.
94/03	L.B. Hartman K.M. van Hee	Application of Marcov Decision Processe to Search Problems, p. 21.
94/04	J.C.M. Baeten J.A. Bergstra	Graph Isomorphism Models for Non Interleaving Process Algebra, p. 18.
94/05	P. Zhou J. Hooman	Formal Specification and Compositional Verification of an Atomic Broadcast Protocol, p. 22.
94/06	T. Basten T. Kunz J. Black M. Coffin D. Taylor	Time and the Order of Abstract Events in Distributed Computations, p. 29.
94/07	K.R. Apt R. Bol	Logic Programming and Negation: A Survey, p. 62.
94/08	O.S. van Roosmalen	A Hierarchical Diagrammatic Representation of Class Structure, p. 22.
94/09	J.C.M. Baeten J.A. Bergstra	Process Algebra with Partial Choice, p. 16.
94/10	T. verhoeff	The testing Paradigm Applied to Network Structure. p. 31.
94/11	J. Peleska C. Huizing C. Petersohn	A Comparison of Ward & Mellor's Transformation Schema with State- & Activitycharts, p. 30.
94/12	T. Kloks D. Kratsch H. Müller	Dominoes, p. 14.
94/13	R. Seljée	A New Method for Integrity Constraint checking in Deductive Databases, p. 34.
94/14	W. Peremans	Ups and Downs of Type Theory, p. 9.
94/15	R.J.M. Vaessens E.H.L. Aarts J.K. Lenstra	Job Shop Scheduling by Local Search, p. 21.
94/16	R.C. Backhouse H. Doornbos	Mathematical Induction Made Calculational, p. 36.
94/17	S. Mauw M.A. Reniers	An Algebraic Semantics of Basic Message Sequence Charts, p. 9.

94/18	F. Kamareddine R. Nederpelt	Refining Reduction in the Lambda Calculus, p. 15.
94/19	B.W. Watson	The performance of single-keyword and multiple-keyword pattern matching algorithms, p. 46.
94/20	R. Bloo F. Kamareddine R. Nederpelt	Beyond β-Reduction in Church's $\lambda\rightarrow$, p. 22.
94/21	B.W. Watson	An introduction to the Fire engine: A C++ toolkit for Finite automata and Regular Expressions.
94/22	B.W. Watson	The design and implementation of the FIRE engine: A C++ toolkit for Finite automata and regular Expressions.
94/23	S. Mauw and M.A. Reniers	An algebraic semantics of Message Sequence Charts, p. 43.
94/24	D. Dams O. Grumberg R. Gerth	Abstract Interpretation of Reactive Systems: Abstractions Preserving ∀CTL*, ∃CTL* and CTL*, p. 28.
94/25	T. Kloks	$K_{1,3}$-free and W_4-free graphs, p. 10.
94/26	R.R. Hoogerwoord	On the foundations of functional programming: a programmer's point of view, p. 54.
94/27	S. Mauw and H. Mulder	Regularity of BPA-Systems is Decidable, p. 14.
94/28	C.W.A.M. van Overveld M. Verhoeven	Stars or Stripes: a comparative study of finite and transfinite techniques for surface modelling, p. 20.
94/29	J. Hooman	Correctness of Real Time Systems by Construction, p. 22.
94/30	J.C.M. Baeten J.A. Bergstra Gh. Ştefanescu	Process Algebra with Feedback, p. 22.
94/31	B.W. Watson R.E. Watson	A Boyer-Moore type algorithm for regular expression pattern matching, p. 22.
94/32	J.J. Vereijken	Fischer's Protocol in Timed Process Algebra, p. 38.