A correspondence between Nuprl and the Ramified Theory of Types

Citation for published version (APA):

Document status and date:
Published: 01/01/1996

Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 02. Oct. 2020
A Correspondence between Nuprl and the Ramified Theory of Types

by

Fairouz Kamareddine and Twan Laan

ISSN 0926-4515

All rights reserved
editors: prof.dr. R.C. Backhouse
         prof.dr. J.C.M. Baeten

Reports are available at:
http://www.win.tue.nl/win/cs

Computing Science Report 96/12
Eindhoven, May 1996
Abstract

In Russell's Ramified Theory of Types (RTT), two hierarchical concepts dominate: orders and types. The use of orders has as a consequence that the logic part of RTT is predicative. The concept of order however, is almost dead since Ramsey eliminated it from RTT. This is why we find Church's simple theory of types (which uses the type concept without the order one) at the bottom of the Barendregt Cube rather than RTT. Despite the disappearance of orders which have a strong correlation with predicativity, predicative logic still plays an influential role in Computer Science. An important example is the proof checker Nuprl, which is based on Martin-Löf's Type Theory which uses type universes. Those type universes, and also degrees of expressions in AUTHOMATH, are closely related to orders. In this paper, we concentrate on Nuprl. We describe Nuprl as a Pure Type System and relate it to Russell's RTT. To be successful at so doing, we are forced to explicitly typing Nuprl (but this is not restrictive). We show also that Russell's orders play a crucial role in understanding the hierarchy of Nuprl.

1 Introduction

The Ramified Theory of Types (RTT) was developed by Bertrand Russell [21, 24] in order to solve the paradoxes that resulted from Frege's "Grundgesetze der Arithmetik" [6]. It has a double hierarchy: one of types (which can be seen as an elementary version of Church's well-known Simple Theory of Types [2]) and one of orders, which can be compared with Kripke's Hierarchy of Truths, see [13, 11]. The hierarchy of orders is less known, as it became unpopular when Ramsey [19] and Hilbert and Ackermann [7] showed that one can avoid the paradoxes without this hierarchy. Nevertheless, orders can elegantly explain some useful hierarchies. In [11], for example, we showed that RTT's orders correspond to the hierarchy of truths where each level corresponds to Tarski's truth notion, but the limit corresponds to a level which can talk about its own truth. Orders moreover, are still present in Martin-Löf's philosophical conceptions of type theories and especially in his layers of universes [16], as we shall illustrate in this paper (via Nuprl [3]). Furthermore, orders are closely related to the degree of expression notion of AUTHOMATH [18].

Logic based on the double hierarchy of orders and types is usually called predicative. The difference between predicative and impredicative logic may seem small, nevertheless, this small difference can have some drastic consequences in fundamental mathematics. When constructing the real numbers out of the rationals (with Dedekind-cuts), the Theorem of the Lowest Upper Bound\(^1\), is not provable in predicative logic (see [23]). The Theorem of the Lowest Upper Bound is, however, one of the most fundamental theorems in real analysis.

---

\(^1\)This work is partially supported by EPSRC grant GR/K 25014. The authors are grateful for some useful discussions with Roel Bloo, Herman Geuvers and Jan Zwanenburg.

1Department of Computing Science, University of Glasgow, 17 Lilybank Gardens, GLASGOW G12 8QQ, SCOTLAND, e-mail: faeiralou@dcs.gla.ac.uk, tel. +44 141 3306042 - fax +44 141 3304913.

1Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O.Box 513, 5600 MB EINDHOVEN, THE NETHERLANDS, e-mail: laan@win.tue.nl, tel. +31 40 2475004 - fax +31 40 2463992. Laan is supported by the Co-operation Centre Tilburg and Eindhoven Universities. He is grateful to the Department of Computing Science, University of Glasgow, for their hospitality and financial support when preparing this article.

1This Theorem states that any non-empty set of real numbers with an upper bound has a lowest upper bound.
Many modern type systems are impredicative. For instance, the systems of the Barendregt cube [1] that have the rule "(\xi,\delta)" are all impredicative. Hence, a proof checker like Coq [5], based on the Calculus of Constructions [4], is itself founded on impredicative logic.

Nevertheless, mathematics with predicative logic is possible, and from a constructive point of view it is even attractive. For instance, the proof checker Nuprl [3, 10] is based on predicative logic yet many mathematical theories can be developed using this proof checker (see [9]).

Nuprl's type theory is related to type theories proposed by Martin-Löf [16], used as a foundation for constructive mathematics. Nuprl's logic is related to its type theory via the well-known propositions-as-types embedding, also known as the Curry-Howard-de Bruijn isomorphism (see [8]). It is constructive on two points: It is based on intuitionistic logic (as is the Curry-Howard-de Bruijn isomorphism) and it is based on predicative logic.

In Section 2 we give a formal description of a part of the type system of Nuprl as a Pure Type System (PTS) [22]. The systems of the Barendregt cube are examples of PTSs. Nuprl in PTS style enables us to formalize the concept of order in Nuprl and to show its correctness. This order classifies types and terms of Nuprl into their relevant hierarchy.

Before we can give any correspondence between RTT and Nuprl, we must give a formal presentation of all of Nuprl's type constructors, as we will only need parts of it. The general, the types) of (A, B)) are the orders in RTT correspond to orders in Nuprl.

In Section 3 we present an embedding of RTT in Nuprl's type system. As we present Nuprl within the framework of PTSs, we also obtain a description of RTT in PTS-style. We show that the orders in RTT correspond to orders in Nuprl.

2 The Nuprl type system

2a A fragment of Nuprl in PTS-style

We give a description of a part of the type system on which Nuprl is based (see [9, 3]). We don't give a full presentation of all of Nuprl's type constructors, as we will only need parts of it. The description of the typing rules is given in a natural deduction style similar to that used in the Barendregt Cube [1], and Pure Type Systems [22].

Below we assume V to be a set of variables, \( \mathbb{Z} \) to be the set of integers, and \( \mathbb{S} = \{ *_1, *_2, \ldots \} \) a set of sorts. The intuition behind the sort \(*_a\) is that it represents the propositions (and, more general, the types) of order \( \leq a \). \(*_a\) corresponds to the Universe of Types \( \mathbb{U}_a \) in [9, 16].

**Definition 2.1** (Terms) The set of terms \( \mathbb{T} \) is defined by the following abstract syntax:
\[
\mathbb{T} ::= \mathbb{S} | V | \perp | Z | TT | \lambda V . T | \Pi V : T . T | T \times T | (T, T) | \tau_1(T) | \tau_2(T)
\]

We let \( \alpha, \beta, x, y, z, \ldots \) range over \( V \); \( m, n, \ldots \) over \( \mathbb{Z} \) and \( A, B, M, N, a, b \) over \( \mathbb{T} \). When \( x \) does not occur free in \( B \), we write \( A \rightarrow B \) for \( \Pi x : A . B \). Free and bound variables are defined as usual.

\( \text{fv}(A) \) and \( \text{bv}(A) \) denote the set of free and bound variables of \( A \). \( A[x:=B] \) denotes the term in which all the free occurrences of \( x \) in \( A \) have been replaced by \( B \). Syntactic equality of terms is taken modulo renaming of bound variables. This allows us to assume the following:

**Convention 2.2** (Barendregt's Convention) Names of bound variables differ from the free ones in a term. Moreover, we use different bound names for different bound variables.

We take the axiom \((\rightarrow) : (\lambda x : T . A) B \rightarrow \beta A[x:=B] \) and the axiom \((\rightarrow) : \tau_1((A, B)) \rightarrow \beta A \) and \( \tau_2((A, B)) \rightarrow \beta B \). We define the reduction relations \( \rightarrow_b \) and \( \rightarrow_\beta \) generated by the above two axioms respectively (with the usual compatibility rules of course). \( \rightarrow_b \) and \( \rightarrow_\beta \) are the reflexive transitive closures of \( \rightarrow \) and \( \beta \). We define moreover \( \rightarrow_\beta \) and \( \rightarrow_\beta \) in the obvious way and take \( \rightarrow_\beta \) to be the symmetric closure of \( \rightarrow_\beta \).

**Definition 2.3** (Contexts) A context is a finite list \( \tau_1 : A_1, \ldots, \tau_n : A_n \) of declarations \( \tau_i : A_i \). \( \{ \tau_1, \ldots, \tau_n \} \) is called the domain of the context. If \( \Gamma, \Delta \) are contexts then we write \( \Gamma \subseteq \Delta \) if all declarations in \( \Gamma \) are also in \( \Delta \). We let \( \Gamma, \Delta \) range over contexts.
Definition 2.4 (Derivable statements) A statement \( \Gamma \vdash A : B \) is derivable if it can be deduced by repeated application of the rules below:

(Axioms)  
\[
\begin{align*}
\Gamma \vdash \bot &: *_1 \\
\Gamma \vdash Z : *_1 \\
\Gamma \vdash n : Z \quad (n \in \mathbb{N}) \\
\end{align*}
\]

(Start)  
\[
\begin{align*}
\Gamma \vdash A : *_n \\
\Gamma, z : A \vdash z : A \\
\Gamma, z : A \vdash M : N \\
\Gamma \vdash A : *_n \\
\Gamma, z : A, x : B \vdash x : B \\
\Gamma \vdash (\lambda x : A. B) : *_n \\
\end{align*}
\]

(Weak)  
\[
\begin{align*}
\Gamma \vdash (\Pi x : A. B) : *_n \\
\Gamma, z : A, x : B \vdash b : B \\
\Gamma \vdash (\Pi x : A. B) : *_n \\
\end{align*}
\]

(II-form)  
\[
\begin{align*}
\Gamma \vdash A : *_n \\
\Gamma, x : A \vdash B : *_n \\
\end{align*}
\]

(Left)  
\[
\begin{align*}
\Gamma \vdash M : (A \times B) \\
\Gamma \vdash \pi_1(M) : A \\
\end{align*}
\]

(Right)  
\[
\begin{align*}
\Gamma \vdash M : (A \times B) \\
\Gamma \vdash \pi_2(M) : B \\
\end{align*}
\]

(Pairs)  
\[
\begin{align*}
\Gamma \vdash a : A \\
\Gamma \vdash b : B \\
\Gamma \vdash (a, b) : (A \times B) \\
\end{align*}
\]

(App)  
\[
\begin{align*}
\Gamma \vdash M : N \vdash B[x := N] \\
\Gamma \vdash A : *_n \\
\Gamma \vdash B : *_n \\
\end{align*}
\]

(x-form)  
\[
\begin{align*}
\Gamma \vdash (a \times B) : *_n \\
\Gamma \vdash A : *_n \\
\Gamma \vdash B : *_n \\
\end{align*}
\]

(\( \subseteq \))  
\[
\begin{align*}
\Gamma \vdash M : A \\
\Gamma \vdash B : *_n \\
A \Rightarrow_{\beta} B \\
\end{align*}
\]

Definition 2.5

- \( \Gamma \) is called legal if there are \( A, B \) such that \( \Gamma \vdash A : B \);
- \( A \) is called legal if there are \( \Gamma, B \) such that \( \Gamma \vdash A : B \) or \( \Gamma \vdash B : A \);
- \( A \) is called a \( \Gamma \)-term if there is \( B \) such that \( \Gamma \vdash A : B \) or \( \Gamma \vdash B : A \);
- \( A \) is called a \( \Gamma \)-type if there is \( n \) such that \( \Gamma \vdash A : *_n \).

We now show some PTS properties of the Nuprl type system.\(^2\) Omitted proofs are as in [1].

Theorem 2.6 (Church Rosser Theorem for \( \rightarrow_{\beta} \) and \( \rightarrow_{\sigma} \))

1. If \( A \rightarrow_{\beta} B_1 \) and \( A \rightarrow_{\beta} B_2 \) then there is \( C \) such that \( B_1 \rightarrow_{\beta} C \) and \( B_2 \rightarrow_{\beta} C \).
2. If \( A \rightarrow_{\sigma} B_1 \) and \( A \rightarrow_{\sigma} B_2 \) then there is \( C \) such that \( B_1 \rightarrow_{\sigma} C \) and \( B_2 \rightarrow_{\sigma} C \).

Proof: 2: any orthogonal term rewrite system (hence \((\mathcal{T}, \rightarrow_{\beta})\)) is Church Rosser (see [12]). \(\square\)

Theorem 2.7 (Church Rosser Theorem for \( \rightarrow_{\beta\sigma} \))

1. If \( A \rightarrow_{\beta} B_1 \) and \( A \rightarrow_{\sigma} B_2 \) then \( \exists C \) such that \( B_1 \rightarrow_{\sigma} C \) and either \( B_2 \rightarrow_{\beta} C \) or \( B_2 \equiv C \);
2. If \( A \rightarrow_{\beta} B_1 \) and \( A \rightarrow_{\sigma} B_2 \) then \( \exists C \) such that \( B_1 \rightarrow_{\sigma} C \), and either \( B_2 \rightarrow_{\beta} C \) or \( B_2 \equiv C \);
3. If \( A \rightarrow_{\beta} B_1 \) and \( A \rightarrow_{\sigma} B_2 \) then \( \exists C \) such that \( B_1 \rightarrow_{\sigma} C \) and \( B_2 \rightarrow_{\beta} C \);

\(^2\)We remark that we presented an explicitly typed (with terms of the form \( \lambda x : A. B \)) version of Nuprl, whilst Nuprl itself is implicitly typed (with terms of the form \( \lambda x : B \)).
4. \( \rightarrow_{\beta \sigma} \) has the Church Rosser property.

**Proof:** Induction on the structure of \( A \). 2: use 1. 3: use 2. 4: use 3 and Theorem 2.6. \( \square \)

**Lemma 2.8 (Transitivity Lemma)** Let \( \Gamma, \Delta \) be legal contexts such that \( \Gamma \vdash x : C \) for all \( x : C \in \Delta \). Then \( \Delta \vdash A : B \Rightarrow \Gamma \vdash A : B \).

**Lemma 2.9 (Substitution Lemma)** If \( \Gamma, \Delta x : A \vdash B \) and \( \Gamma \vdash D : A \) then \( \Delta; x := D \vdash B[x := D] : C[x := D] \).

**Lemma 2.10 (Thinning Lemma)** Let \( \Gamma, \Delta \) be legal contexts, \( \Delta \subseteq \Delta \). \( \Gamma \vdash A : B \Rightarrow \Delta \vdash A : B \).

**Lemma 2.11 (Generation Lemma)**
1. If \( \Gamma \vdash *_n : C \) then \( C \equiv \beta \sigma *_m \) for some \( m > n \), and if \( C \not\equiv *_m \) then \( \Gamma \vdash : *_p \) for some \( p \geq 1 \).
2. If \( \Gamma \vdash \bot : C \) then \( C \equiv \beta \sigma *_m \) for some \( m \geq 1 \), and if \( C \not\equiv *_m \) then \( \Gamma \vdash \bot : *_p \) for some \( p \geq 1 \).
3. If \( \Gamma \vdash x : C \) then \( C \equiv \beta \sigma *_m \) for some \( m \geq 1 \), and if \( C \not\equiv *_m \) then \( \Gamma \vdash x : *_p \) for some \( p \geq 1 \).
4. If \( \Gamma \vdash \gamma(A \vdash B) : C \) then \( C \equiv \beta \sigma *_m \), and if \( C \not\equiv *_m \) then \( \Gamma \vdash \gamma(A \vdash B) : *_p \) for some \( p \geq 1 \).
5. If \( \Gamma \vdash x : C \) then there is \( B \) such that \( x : B \in \Gamma \), and either \( B \equiv \beta \sigma *_m \), or there are \( m, n \) with \( m < n \) and \( B \equiv \beta \sigma *_m \). If \( C \not\equiv B \) then \( \Gamma \vdash x : *_p \) for some \( p \geq 1 \).
6. If \( \Gamma \vdash (\Pi x : A. B) : C \) then there is \( m \) such that \( \Gamma \vdash A : *_m \), \( \Gamma, x : A \vdash B : *_m \) and \( C \equiv \beta \sigma *_n \) for some \( n \geq m \). If \( C \not\equiv *_n \) then \( \Gamma \vdash : *_p \) for some \( p \geq 1 \).
7. If \( \Gamma \vdash (\lambda x : A. B) : C \) then there are \( B, C \) such that \( \Gamma \vdash (\Pi x : A. B) : *_m \), \( \Gamma, x : A \vdash B : *_m \) and \( C \equiv \beta \sigma *_n \). If \( C \not\equiv *_n \) then \( \Gamma \vdash : *_p \) for some \( p \geq 1 \).
8. If \( \Gamma \vdash AB : C \) then there are \( x, P, Q \) such that \( \Gamma \vdash A : (\Pi x : P. Q) \), \( \Gamma \vdash B : P \) and either \( C \equiv \beta \sigma Q[x := B] \), or there are \( m, n \) with \( m < n \) and \( Q[x := B] \equiv \beta \sigma *_m \) and \( C \equiv \beta \sigma *_n \). If \( C \not\equiv Q[x := B] \) then \( \Gamma \vdash : *_p \) for some \( p \geq 1 \).
9. If \( \Gamma \vdash (A \times B) : C \) then there is \( m \) such that \( \Gamma \vdash A : *_m \), \( \Gamma \vdash B : *_m \) and \( C \equiv \beta \sigma *_n \) for some \( n \geq m \). If \( C \not\equiv *_n \) then \( \Gamma \vdash : *_p \) for some \( p \geq 1 \).
10. If \( \Gamma \vdash (a, b) : C \) then there are \( a, b \) such that \( \Gamma \vdash (A \times B) : *_m \), \( \Gamma \vdash a : A \), \( \Gamma \vdash b : B \), and \( C \equiv \beta \sigma A \times B \). If \( C \not\equiv A \times B \) then \( \Gamma \vdash : *_p \) for some \( p \geq 1 \).
11. If \( \Gamma \vdash \pi_1(M) : C \) then there are \( A_1, A_2 \) such that \( \Gamma \vdash M : (A_1 \times A_2) \) and either \( C \equiv \beta \sigma A_1 \) or there are \( m, n \) with \( m < n \) and \( A_i \equiv \beta \sigma *_n \) and \( C \equiv \beta \sigma *_n \).

**Proof:** Tedious but straightforward induction on the derivation \( \Gamma \vdash M : C \). \( \square \)

**Corollary 2.12 (Correctness of Types)** If \( \Gamma \vdash A : B \) then there is \( n \geq 1 \) such that \( \Gamma \vdash B : *_n \).

**Theorem 2.13 (Subject Reduction)** If \( \Gamma \vdash A : B \) and \( A \rightarrow_{\beta \sigma} A' \) then \( \Gamma \vdash A' : B \).

Due to \( \subseteq \), Unicity of Types doesn’t hold for Nuprl. A weak version however, is possible:

**Definition 2.14** For each term \( A \) we define a term \( \overline{A} \) as follows:

\[
\begin{align*}
| *_m | &= *_1 ; \\
| \Pi x : A. B | &= \Pi x : A. | B | ; \\
| x | &= x ; \\
| \bot | &= \bot ; \\
| C | &= C ; \\
| (A, B) | &= | A | \times | B | ; \\
| \lambda x : A. B | &= \lambda x : A. | B | ; \\
| M \| N | &= | M | \| N | ; \\
| \pi_1(M) | &= \pi_1(| M |) ; \\
| \pi_2(M) | &= \pi_2(| M |)
\end{align*}
\]

**Theorem 2.15 (Weak Unicity of Types)** If \( \Gamma \vdash A : B_1 \) and \( \Gamma \vdash A : B_2 \) then \( | B_1 | \equiv \beta \sigma | B_2 | \).

**Proof:** Induction on the structure of \( A \). We only treat \( A \equiv (\lambda x : M. N) \). By Lemma 2.11, \( \exists D_1, D_2 \) with \( B_1 \equiv \beta \sigma \Pi x : M. D_1 \), and \( \Gamma, x : M \vdash N : D_2 \). By the induction hypothesis, \( | D_1 | \equiv \beta \sigma | D_2 | \). Hence, \( | B_1 | \equiv \beta \sigma | \Pi x : M. D_1 | \equiv \Pi x : M. | D_1 | \equiv \beta \sigma \Pi x : M. | D_2 | \equiv | \Pi x : M. D_2 | \equiv \beta \sigma | B_2 | \). \( \square \)
2b Orders in Nuprl

Correctness of Types makes the following lemma and definition possible:

Lemma 2.16 If A is a Γ-term then ∃ a Γ-term B, ∃ n ≥ 1 such that Γ ⊢ A : B : *n.

Proof: A is a Γ-term ⇒ ∃ Γ-term B with Γ ⊢ A : B or Γ ⊢ B : A. If Γ ⊢ A : B, then by Correctness of Types ∃ n ≥ 1 where Γ ⊢ A : B : *n. If Γ ⊢ B : A then again by Correctness of Types ∃ n ≥ 1 where Γ ⊢ A : *n and hence by Start and Thinning, Γ ⊢ A : *n : *n+1. □

Note that if Γ is a Γ-term then for any A' where A →βσ A', A' is a Γ-term. There are also A' =βσ A where A →βσ A' yet A' is a Γ-term. We define [A]r = {A' | A' is Γ-term and A =βσ A'}. Now, we define the order of a term:

Definition 2.17 Assume A is a Γ-term. We define ordr(A), the order of A in Γ, as the smallest natural number a for which there are A' ∈ [A]r and B such that Γ ⊢ A' : B : *n+1.

We prove some elementary properties of ordr(A):

Lemma 2.18 Let A be a Γ-term and ordr(A) = a. The following holds:

1. If A' ∈ [A]r then ordr(A') = ordr(A).
2. There are A' and B such that Γ ⊢ A' : B : *n and A →βσ A'.

Proof: 1: easy. 2: by definition of ordr(A), ∃ A'' =βσ A and B where Γ ⊢ A'' : B : *n+1. By Church Rosser, A, A'' have a common reduct, say A'. By Subject Reduction, Γ ⊢ A' : B : *n+1. □

Corollary 2.19 For a Γ-term (A, B), ordr(π(1((A, B)))) = ordr(A) and ordr(π(2((A, B)))) = ordr(B).

Corollary 2.20 For a Γ-term A in βσ-normal form and ordr(A) = a, ∃ B where Γ ⊢ A : B : *n+1.

The order of a term does not change if the context is expanded:

Lemma 2.21 If Γ ⊢ A : B and Γ, x : C is legal, then ordr(A) = ordr, x : C η (A).


Corollary 2.22 If Γ ⊢ A is a Γ-term and Γ ⊢ Δ ≥ Γ is legal then ordr(Δ) = ordr(Δ).

The order of a term does not increase under substitution:

Lemma 2.23 If Γ, x : A, Δ ⊢ B : C and Γ ⊢ D : A then ordr, x : A, Δ η (B) ≥ ordr, x : A, Δ η (D).


ordr, x : A, Δ η (B) = ordr, x : A, Δ η (D) does not hold in general: take Γ ⊢ y : a. Then Γ ⊢ x := x + 2 and Γ ⊢ y : x + 2, and by Lemma 2.28 below) ordr(x := y) = 2 and ordr(x := y) = ordr(y) = 1.

The orders of constants and sorts are easy to calculate:

Lemma 2.24 (Orders of constants and sorts) Let Γ be a legal context. Then ordr(*a) = a + 1, ordr(⊥) = 1, ordr(1) = 1, and ordr(0) = 0.

Proof: We only prove ordr(*a) = a + 1 (the other cases are similar). As Γ ⊢ *a : *n+1 : *n+2, ordr(*a) ≤ a + 1. Now assume Γ ⊢ A' : *a for an A' =βσ *a (hence A' =βσ *a). By repeated Subject Reduction, Γ ⊢ A : *a. By Generation, P =βσ *c for a c > a (hence P = βσ *c). By repeated Subject Reduction, Γ ⊢ *a : *a, so again by Generation, 3d > c where s = βσ *d. Hence d = b, so a < c < b, so b ≥ a + 2, so ordr(*a) ≥ a + 1. □

5
If \( B \) is a \( \Gamma \)-type then there is always \( B' \) of type \( \ast_{\text{ord}_\Gamma(B)} \) such that \( B \rightarrow_{\beta_{\ast_B}} B' \):

**Lemma 2.25** Let \( B \) be a \( \Gamma \)-type and \( b = \text{ord}_\Gamma(B) \). \( \exists B' \) such that \( \Gamma \vdash B' : *_b \) and \( B \rightarrow_{\beta_{\ast_B}} B' \).

**Proof:** Assume \( \Gamma \vdash B : *_p \). By Lemma 2.18, \( \exists B' \) and \( P \) such that \( \Gamma \vdash B' : *_{b+1} \) and \( B \rightarrow_{\beta_{*b}} B' \). By Weak Unicity of Types 2.15, \( \beta(P) \vdash \beta(P) \), i.e., \( P \equiv_{\beta_{*b}} q \). Hence \( P \equiv_{\beta_{*b}} *_q \).

- By repeated Subject Reduction, \( \Gamma \vdash *_q : *_{b+1} \). By Lemma 2.24, \( b + 1 \geq q + 1 \), so \( b \geq q \).
- By the Conversion Rule, \( \Gamma \vdash B' : *_q : *_{q+1} \), so by definition of \( b \) and \( q \).

We find: \( q = b \), so \( P \equiv_{\beta_{*b}} *_q \), so by conversion \( \Gamma \vdash B' : *_b \).

This lemma confirms that \( *_a \) can be seen as the type of types (propositions) of order \( \leq a \):

**Corollary 2.26** If \( P \) is a \( \Gamma \)-type in \( \beta_{a}\)-normal form, then \( \Gamma \vdash P : *_a \iff \text{ord}_\Gamma(P) \leq a \).

**Proof:** Let \( p = \text{ord}_\Gamma(P) \). \( \Rightarrow \) is by definition of \( \text{ord}_\Gamma(P) \); for \( \Leftarrow \), by Lemma 2.25, \( \exists P' \) where \( \Gamma \vdash P' : *_p \) and \( P \rightarrow_{\beta_{*p}} P' \). As \( P \) is in normal form, \( P' \equiv P \), so \( \Gamma \vdash P : *_p \). Since \( p \leq a \), repeated use of \((\subseteq)\) derives \( \Gamma \vdash P : *_a \).

Another result is that a term is always of a lower order than its type:

**Corollary 2.27** If \( \Gamma \vdash A : B \) then \( \text{ord}_\Gamma(A) < \text{ord}_\Gamma(B) \).

**Proof:** Let \( a = \text{ord}_\Gamma(A) \), \( b = \text{ord}_\Gamma(B) \). \( B \) is a type, so by Lemma 2.25, \( \exists B' \) where \( \Gamma \vdash B' : *_b \) and \( B \rightarrow_{\beta_{*b}} B' \). \( \Gamma \vdash A : B \), so by conversion, \( \Gamma \vdash A : B' : *_b \). By definition of \( a \), \( b \geq a + 1 \), so \( b > a \).

In the above corollary, \( \text{ord}_\Gamma(A) = \text{ord}_\Gamma(B) - 1 \) does not hold: take \( \Gamma = \emptyset \), \( A \equiv *_1 \), and \( B \equiv *_2 \).

In Lemma 2.24, we calculated the order of sorts and constants. Now, we present methods to calculate the order of almost all the other terms:

**Lemma 2.28** Let \( C \) be a \( \Gamma \)-term. The following holds:

1. If \( C \equiv x \) where \( x : A \in \Gamma \) then \( \text{ord}_\Gamma(x) = \text{ord}_\Gamma(A) - 1 \).
2. If \( C \equiv \Pi x : A. B \) then \( \text{ord}_\Gamma(\Pi x : A. B) = \max(\text{ord}_\Gamma(A), \text{ord}_\Gamma x : A(B)) \).
3. If \( C \equiv \lambda x : A. B \) then \( \text{ord}_\Gamma(\lambda x : A. B) = \max(\text{ord}_\Gamma(A), \text{ord}_\Gamma x : A(B)) \).
4. If \( C \equiv A \times B \) or \( C = (A, B) \) then \( \text{ord}_\Gamma(C) = \max(\text{ord}_\Gamma(A), \text{ord}_\Gamma(B)) \).

**Proof:** 1: Let \( m = \text{ord}_\Gamma(x) \). From Corollary 2.20, \( \exists B \) with \( \Gamma \vdash x : B : *_{m+1} \). As \( m + 1 \) is minimal, \( \text{ord}_\Gamma(B) = m + 1 \). By the Generation Lemma, \( A \equiv_{\beta_{*B}} B \). Hence, \( \text{ord}_\Gamma(A) = m + 1 \).

2: Let \( a = \text{ord}_\Gamma(A) \), \( b = \text{ord}_\Gamma x : A(B) \), and \( p = \text{ord}_\Gamma(\Pi x : A. B) \). By Lemma 2.25, \( \exists P \) with \( \Gamma \vdash P : *_p \) and \( \Pi x : A. B \rightarrow_{\beta_{*p}} P \). \( P \) must be of the form \( \Pi x : A. B_1 \), where \( A \rightarrow_{\beta_{*a}} A_1 \) and \( B \rightarrow_{\beta_{*B}} B_1 \). By Lemmas 2.25 and 2.11, \( \exists A_2 \) and \( B_2 \) such that \( \Gamma \vdash A_2 : *_a \), \( \Gamma \vdash x : A_1 \rightarrow B_2 : *_{b_2} \), \( A \rightarrow_{\beta_{*a}} A_2 \) and \( B \rightarrow_{\beta_{*B}} B_2 \). By Church Rosser, \( A_1 \) and \( A_2 \) have a common reduct \( A_3 \); \( B_1 \) and \( B_2 \) have a common reduct \( B_3 \). By repeated Subject Reduction: \( \Gamma \vdash A_3 : *_{a} \), \( \Gamma \vdash x : A_3 \rightarrow B_3 : *_{b} \). As \( A \rightarrow_{\beta_{*a}} A_3 \) and \( B \rightarrow_{\beta_{*B}} B_3 \), Subject Reduction gives \( \Gamma \vdash (\Pi x : A_3.B_3) : *_{*} \). Now, \( p = \text{ord}_\Gamma(a, b) \) as follows:

- By Generation \( \exists m \leq p \) with \( \Gamma \vdash A_3 : *_{m} \) and \( \Gamma \vdash x : A_3 \rightarrow B_3 : *_{m} \). By Transitivity, \( \Gamma \vdash x : A \rightarrow B_3 : *_{m} \). Hence \( a, b \leq m \).
- As \( \Gamma \vdash A_3 : *_{a} \) and \( \Gamma \vdash x : A_3 \rightarrow B_3 : *_{b} \), so by repeated application of \((\subseteq)\), \( \Gamma \vdash A_3 : *_{\max(a, b)} \) and \( \Gamma \vdash x : A_3 \rightarrow B_3 : *_{\max(a, b)} \). By II-formation, \( \Gamma \vdash (\Pi x : A_3.B_3) : *_{\max(a, b)} \), and so \( p \leq \max(a, b) \).
3: Let \( a = \text{ord}_\Gamma(A) \), \( m = \text{ord}_\Gamma(\lambda x : A. B) \), \( n = \text{ord}_\Gamma x : A(B) \). By Lemma 2.18, \( \exists P \) where \( \Gamma \vdash P : Q : *_{m+1} \) and \( \lambda x : A. B \rightarrow_{\beta_{*b}} P \). Observe that \( P \equiv \lambda x : A'. B' \) for some \( A', B' \) with \( A \rightarrow_{\beta_{*a}} A' \) and \( B \rightarrow_{\beta_{*B}} B' \). By the Generation Lemma, \( \exists B \) such that \( \Gamma \vdash x : A' \rightarrow B' : *_{n} \). Now \( m + 1 = \text{ord}_\Gamma(\Pi x : A'. B') = \text{ord}_\Gamma(\Pi x : A. B) = \max(a, \text{ord}_\Gamma x : A(B)) \) by 2 above. Now \( m = \max(a, n + 1) \) because \( m + 1 = \max(a, n + 1) \) as is seen by the two cases:
The Ramified Theory of Types RTT

In this section we give a short, formal description of Russell’s Ramified Theory of Types (RTT). This formalisation is both faithful to Russell’s original informal presentation and compatible with the present formulations of type theories. The basic aim of RTT is to exclude the logical paradoxes from logic by eliminating all self-references. An extended philosophical motivation for this theory can be found in [24], pages 38-55. We will not go into the full details of the formalisation of RTT (these details can be found in [14], the presentation by Russell himself in “Principia” is informal).

In Subsection 3a we introduce propositional functions. In Subsection 3b we assign types to some of these propositional functions. Paradoxical propositional functions are, of course, not typeable.

3a Propositional Functions

In this section we shall describe the set of propositions and propositional functions which Whitehead and Russell use in “Principia”. We give a modernised, formal definition which corresponds to the description in “Principia”. At the basis of the system of our formalization there is

- an infinite set $A$ of individual-symbols and an infinite set $V$ of variables;
- an infinite set $R$ of relation-symbols together with an arity map $a : R \to \mathbb{N}^+$.

0-ary relations are not explicitly used in “Principia” but could be added without problems. Since functions are relations in Principia, we will not introduce a special set of function symbols.

We assume that \( \{ a_1, a_2, \ldots \} \subseteq A \); \( \{ x, x_1, x_2, \ldots, y, y_1, \ldots, z, z_1, \ldots \} \subseteq V \); \( \{ R, R_1, \ldots, S, S_1, \ldots \} \subseteq R \). We will use the letters $x, y, z, x_1, \ldots$ as meta-variables over $V$, and $R, R_1, \ldots$ as meta-variables over $R$. Note that variables are written in typewriter style and that meta-variables are written in italics: $x$ denotes one, fixed object in $V$ whilst $x$ denotes an arbitrary object of $V$.

We assume that there is an order (e.g. alphabetical) on the collection $V$, and write $x < y$ if the variable $x$ is ordered before the variable $y$. In particular, we assume that

\[ x < x_1 < \ldots < y < y_1 < \ldots < z < z_1 < \ldots \]
We also have the logical symbols \( \land, \neg \) and \( \forall \) in our alphabet, and the non-logical symbols: parentheses and the comma. Note that Russell used classical logic (intuitionistic logic wasn’t widespread when “Principia” appeared) and hence he didn’t need to make symbols like \( \lor, \rightarrow \), 3 primitive.

**Definition 3.1 (Propositional functions)** We define a collection \( \mathcal{F} \) of propositional functions, and for each element \( f \) of \( \mathcal{F} \) we simultaneously define the collection \( \text{FV}(f) \) of free variables of \( f 

1. If \( R \in \mathcal{R} \) and \( i_1, \ldots, i_{a(R)} \in A \cup \mathcal{V} \) then \( R(i_1, \ldots, i_{a(R)}) \in \mathcal{F} \).  
   \[ \text{FV}(R(i_1, \ldots, i_{a(R)})) \overset{\text{def}}{=} \{i_1, \ldots, i_{a(R)}\} \cap \mathcal{V}; \]
2. If \( z \in \mathcal{V}, n \in \mathbb{N} \) and \( k_1, \ldots, k_n \in A \cup \mathcal{V} \cup \mathcal{F} \), then \( z(k_1, \ldots, k_n) \in \mathcal{F} \).  
   \[ \text{FV}(z(k_1, \ldots, k_n)) \overset{\text{def}}{=} \{z, k_1, \ldots, k_n\} \cap \mathcal{V}. \]
   If \( n = 0 \), we write \( z() \) so as to distinguish the propositional function \( z() \) from the variable \( z \).
3. If \( f, g \in \mathcal{F} \) then \( f \land g \in \mathcal{F} \) and \( \neg f \in \mathcal{F} \).  
   \[ \text{FV}(f \land g) \overset{\text{def}}{=} \text{FV}(f) \cup \text{FV}(g); \text{FV}(\neg f) \overset{\text{def}}{=} \text{FV}(f); \]
4. If \( f \in \mathcal{F} \) and \( z \in \text{FV}(f) \) then \( \forall z[f] \in \mathcal{F} \).  
   \[ \text{FV}(\forall z[f]) = \text{FV}(f) \setminus \{z\}. \]
5. All propositional functions can be constructed by using the rules 1, 2, 3 and 4 above.

We use the letters \( f, g, h \) as meta-variables over \( \mathcal{F} \) and similar to Convention 2.2, we assume that bound variables differ from free ones and that different bound variables have different names.

A propositional function \( f \) is a proposition in which some parts (the free variables) have been left undetermined. It will turn into a proposition as soon as we assign values to all its free variables. In this light, a proposition can be seen as a degenerated propositional function (with 0 free variables).

It will be clear now what the intuition behind propositional function of the form \( R(i_1, \ldots, i_{a(R)}) \), \( f \land g \), \( \neg f \) and \( \forall z[f] \) is. The intuition behind propositional functions of the second kind is not so obvious. \( z(k_1, \ldots, k_n) \) is a propositional function of higher order: \( z \) is a variable for a propositional function with \( n \) free variables; the argument list \( k_1, \ldots, k_n \) indicates what should be substituted\(^4\) for these free variables as soon as one assigns such a propositional function to \( z \).

Notice that there are propositional functions of the form \( z(k_1, \ldots, k_n) \) (where \( z \in \mathcal{V} \)) but that expressions of the form \( f(k_1, \ldots, k_n) \), where \( f \in \mathcal{F} \), are not propositional functions. Even substituting \( f \) for \( z \) in \( z(k_1, \ldots, k_n) \) does not lead to \( f(k_1, \ldots, k_n) \), as the notion of substitution in \( \mathcal{F} \) is quite different from the usual notion of substitution in first order logic.

**Example 3.2** Here are some higher-order propositional functions (pf's) from mathematics:
1. The pf's \( z(x) \) and \( z(y) \) in the definition of Leibniz-equality: \( \forall z[z(x) \rightarrow z(y)] \).
2. The pf's \( z(0) \), \( z(x) \), and \( z(y) \) in the formulation of complete induction:
   \[ [z(0) \rightarrow (\forall x \forall y[z(x) \rightarrow (s(x,y) \rightarrow z(y))])] \rightarrow \forall x[z(x)]. \]
3. The pf \( z() \) in the formulation of the law of the excluded middle: \( \forall z[z() \lor \neg z()] \).

### 3b Ramified Types

Not all propositional functions should be allowed in our language. For instance, the expression \( \neg z(x) \) is a perfectly legal element of \( \mathcal{F} \), nevertheless, it is the propositional function that makes it possible to derive the Russell Paradox. Therefore, types are introduced.

**Definition 3.3 (Ramified Types)** The ramified types \( \mathcal{T} \) are defined inductively as follows:

1. \( 0^0 \) is a ramified type (0 is called the order of this type);
2. If \( t_1, \ldots, t_n \) are ramified types of orders \( a_1, \ldots, a_n \) respectively, and \( a > \max(a_1, \ldots, a_n) \), then \( (t_1, \ldots, t_n)^a \) is a ramified type of order \( a \) (if \( n = 0 \) then take \( a = 1 \));
3. All ramified types can be constructed using the rules 1 and 2.

\(^3\)A variable is not a propositional function. See [20], Chapter VIII: “The variable”, p.94 of the 7th impression.

\(^4\)In Principia, it is not clear how such substitutions are carried out. One must depend on intuition and on how substitution is used in the Principia. It is quite hard and elaborate to give a proper definition of substitution.
\( \ell \) is the type of individuals, and \((t_1, \ldots, t_n)^a\) is the type of the propositional functions with \(n\) free variables, say \(x_1, \ldots, x_n\), such that if we assign values \(k_1\) of type \(t_1\) to \(x_1, \ldots, k_n\) of type \(t_n\) to \(x_n\), then we obtain a proposition. The type \((1)^a\) is the type of propositions of order \(a\).

Russell strictly divides his propositional functions in orders. For instance, both \(\forall p[p() \land \neg p()]\) and \(\forall (a)\) are propositions, but of different level: the first presumes a full collection of propositions, hence it cannot belong to the same collection of propositions as the propositions \(p\) over which it quantifies (among which \(\forall (a)\)). This led Russell to make \(\forall p[p()] \land \neg p()\) belong to a type of a higher order (level) than the order of \(\forall (a)\). This can already be seen in the definition of ramified types: \((t_1, \ldots, t_n)^a\) can only be a type if \(a\) is strictly greater than each of the orders of the \(t_i\).

**Definition 3.4** Let \(x_1, \ldots, x_n\) be a list of distinct variables, and \(t_1, \ldots, t_n\) be a list of ramified types. We call \(x_1: t_1, \ldots, x_n: t_n\) a context and call \(\{x_1, \ldots, x_n\}\) its domain.

We write \(\Gamma \vdash f : t\) to express that \(f \in \mathcal{F}\) has type \(t\) in context \(\Gamma\), and extend the variable convention to contexts: If \(z\) is bound in \(f\), then \(z\) does not occur in the domain of \(\Gamma\).

We use \(\Gamma, \Delta\) to range over contexts and \(t_1, t_2, \ldots\) to range over types. To avoid confusion we sometimes write \(\Gamma_N\) for derivability in the ramified type system, and \(\vdash_R\) for derivability in RTT.

We now present the typing rules for RTT. These rules are derived from and equivalent to the rules in [14], which are as close as possible to Russell's original ideas. We change our notation for propositional functions slightly: Instead of \(x: \ell\) we write \(x: t[f]\), where \(t\) is some ramified type.

**Definition 3.5 (Typing Rules for RTT)**

- If \(c \in A\), then \(\Gamma \vdash c : \ell\) for any context \(\Gamma\).
- If \(f \in \mathcal{F}\), and \(x_1 < \ldots < x_n\) are the free variables of \(f\), and \(t_1, \ldots, t_n\) are types such that \(x_i: t_i \in \Gamma\), then \(\Gamma \vdash f : (t_1, \ldots, t_n)^a\) if and only if
  - If \(f \equiv R(t_1, \ldots, i_{\ell}(R))\) then \(t_i = \ell\) for all \(i\), and \(a = 1\);
  - If \(f \equiv x(k_1, \ldots, k_m)\) then there are \(u_1, \ldots, u_m\) such that \(x(u_1, \ldots, u_m)^{a-1} \in \Gamma\), and \(\Gamma \vdash k_i: u_i\) for all \(k_i \in A \cup \mathcal{F}\), and \(k_i: u_i \in \Gamma\) for all \(k_i \in \mathcal{V}\);
  - If \(f \equiv f_1 \land f_2\) then there are \(u_1, u_2\) such that \(\Gamma \vdash f_1 : u_1^{a_1}\) and \(a = \max(a_1, a_2)\);
  - If \(f \equiv f^\bot\) then \(\Gamma \vdash f' : (t_1, \ldots, t_n)^a\).
  - If \(f \equiv \forall x: t_0[f']\) then there is \(j\) such that \(\Gamma, x: t_0 \vdash f' : (t_1, \ldots, t_{j-1}, t, t_j, \ldots, t_n)^a\).

**Example 3.6** \(\neg x(x)\) is not typeable in any context \(\Gamma\). If \(\Gamma \vdash \neg x(x) : t\) then \(t\) must be of the form \((u)^a\), with \(x: u \in \Gamma\), as \(\neg x(x)\) has one free variable. Hence \(\Gamma \vdash x(x) : (u)^a\), and by Unicity of Types below, \(u \equiv (u')^{a-1}\), with \(x : u' \in \Gamma\). As \(\Gamma\) is a context, \(u \equiv u'\), hence \(u \equiv (u')^{a-1}\). Absurd.

An important result (whose proof follows directly from the definition of \(\Gamma \vdash f : t\)) is the following:

**Theorem 3.7 (Unicity of Types)** if \(\Gamma \vdash f : t\) and \(\Gamma \vdash f : u\) then \(t \equiv u\).

## 4 RTT in Nuprl

We present a straightforward embedding of RTT in the type theory of Nuprl. The embedding will consist of two parts: First we give a representation of the ramified types in Nuprl (Subsection 4a), then we represent the typable propositional functions in Nuprl (Subsection 4b).

### 4a Ramified Types in Nuprl

The main clue to our embedding is the interpretation of \(\ast_n\) as the sort containing all order-\(n\)-propositions. There is a small difference in that Nuprl considers any term of type \(\ast_n\) to be of type \(\ast_{n+1}\) as well. This means that any proposition of order \(n\) can be interpreted as a proposition of order \(n+1\) as well. This inclusion is not a feature of RTT; yet it isn't a serious extension.

Another small point is that Russell doesn't specify his underlying set of "individuals" and that we want to use \(\mathbb{Z}\) as translation of this underlying set. Therefore, we will assume that the set \(A\) of RTT-individuals is equal to the set \(\mathbb{Z}\) of integers.
Definition 4.1 Define a mapping \( T: \mathbb{T} \rightarrow \mathbb{T} \) as follows:

\[
T(0) \overset{\text{def}}{=} Z \quad \text{and} \quad T((t_1^a), \ldots, t_n^a) \overset{\text{def}}{=} T(t_1^a) \rightarrow \ldots \rightarrow T(t_n^a) \rightarrow \ast_a
\]

Note that \( T()^a = \ast_a \) and \( T \) does indeed interpret the type of order-\( a \)-propositions as \( \ast_a \). Moreover, translations of ramified types are typable in Nuprl:

Lemma 4.2 If \( t^a \) is a ramified type of order \( a \) then \( \Gamma_N T(t^a) : \ast_{a+1} \).

Proof: Induction on the construction of ramified types.

When we speak of a ramified type \( t^a \) of order \( a \), we actually mean that the terms of type \( t^a \) have order \( a \). \( T(t^a) \) itself should, therefore, have order \( a + 1 \) in Nuprl. Indeed, we can prove:

Lemma 4.3 If \( r \) is a legal context then \( \text{ordr}(T(t^a)) = a + 1 \).

Proof: Induction on ramified types.

\[
T(\lambda x: t. A) \overset{\text{def}}{=} \lambda x: T(t) \ldots \lambda x: T(t_n). A
\]

The extension \( T \) as defined above also depends on the context \( \Gamma \). Normally it will be clear which context \( \Gamma \) is meant. If confusion arises, we write \( T_\Gamma \) to indicate the context in question.

It is important to notice that, for propositions \( f \), \( T(f) \) is exactly the interpretation of \( f \) provided by the Curry-Howard-de Bruijn isomorphism.

Finally, we define a special Nuprl-context \( \Gamma_0 \) which contains information on the relation \( a(R) \) times \( Z \) individual symbols of RTT by: \( \Gamma_0 \overset{\text{def}}{=} \{ R: \overset{\text{def}}{=} Z \rightarrow \ldots \rightarrow Z \rightarrow t_1 | R \in \mathcal{R} \} \).

We assume \( \mathcal{R} \) to be finite for the moment, so that \( \Gamma_0 \) is finite as well, and therefore is a Nuprl-context. \( \Gamma_0 \) is legal, as we have \( \Gamma_N Z \rightarrow \ldots \rightarrow Z \rightarrow t_1 : \ast_2 \).

The following theorem states that the embedding \( T \) respects the type structure of RTT. This means that we can see Nuprl as an extension of the Ramified Theory of Types.
Theorem 4.5 (Nuprl extends RTT) If $\Gamma \vdash_R f : t$ then $\Gamma_0 \vdash_N T(f) : T(t)$.

Proof: Induction on the definition of $\Gamma \vdash_R f : t$. If $\Gamma \vdash c : c^0$ because $c \in \mathbb{Z}$ then $\mathbb{Z} \in \Gamma_0$, so $\Gamma_0 \vdash c : \mathbb{Z}$. Now assume $f \in F$, $f$ has free variables $z_1 < \ldots < z_n$ and $t_1, \ldots , t_n$ where $z_i: t_i \in \Gamma$ for $i = 1, \ldots , n$, and $\Gamma \vdash_R f : (t_1, \ldots , t_n)^o$. By Lemma 4.2, $\Gamma_0 \vdash T(t_i) : \tau_a$, for some $\tau_a$. Hence, by the Start and Weakening rules, we add $z_i: T(t_i)$ one by one to the context $\Gamma_0$, obtaining a legal context $\Gamma_i = \Gamma_0, z_1: T(t_1), \ldots , z_n: T(t_n)$. We only treat the case $f = \forall x: t_0[x]$:

If $f = \forall x: t_0[x]$ then $f = \forall x: t_0 \Rightarrow \forall \eta : (t_0[x]) \Rightarrow (\forall x: t_0)$. By the induction hypothesis, $\Gamma_0 \vdash (T(t_1) \Rightarrow \ldots \Rightarrow T(t_{i-1}) \Rightarrow T(t_i) \Rightarrow \ldots \Rightarrow T(t_n) \Rightarrow \tau_a)$. By the Generation Lemma, $\Gamma_0, z_1: T(t_1), \ldots , z_{j-1}: T(t_{j-1}), z_j: T(t_j), z: T(t_j), \ldots , z_n: T(t_n) \vdash_N G : \tau_a$ where $g \equiv \lambda x_1 \ldots x_{j-1} x j \ldots x_n. G$. As the types of the variables in the context are independent from each other, we also have $\Gamma_1, z: T(t_0) \vdash_N \tau_a$. As the order of type $t_0$ is smaller than $a$, we have $\Gamma_1 \vdash_N T(t_0) : \tau_a$ (Lemma 4.2), so by II-formation: $\Gamma_1 \vdash_N \Pi x: T(t_0). G : \tau_a$. By $\lambda$-abstracting over all the variables in $FV(f)$ we obtain $\Gamma_0 \vdash_N T(f) : T(t)$.

It would be nice if we could also prove a kind of opposite of Theorem 4.5. However, the statement $\Gamma_0 \vdash_N T(f) : T(t)$ then there is a context $\Gamma$ such that $\Gamma \vdash_R f : t$ is not true. We can derive $\Gamma_0 \vdash_N T(\forall x: c^0[R(x)]) : \tau_a$ for any $n \geq 1$. Nevertheless, we have $\Gamma \vdash_R \forall x: c^0[R(x)] : (\forall x: c^0[R(x)])$ for any $n \geq 1$. It is clear that this difference between RTT and Nuprl is caused by the type inclusion rule $\subseteq$, which is only present in Nuprl, and not in RTT. We do have a partial result, however:

Lemma 4.6 If $\Gamma \vdash_R f : (t_1^o, \ldots , t_n^o)^o$ and $z_1 < \ldots < z_n$ are the free variables of $f$, then $\text{ord}_\Gamma (T(z_1: t_1^o, \ldots , z_n: t_n^o[f])) = a$.

Proof: Induction on the definition of $\Gamma \vdash_R f : (t_1^o, \ldots , t_n^o)^o$. Note that $z_i: t_i^o \in \Gamma$ for all $i$, and $\Gamma \vdash_R \forall x_1: t_1^o \ldots \forall x_n: t_n^o[f] : (\forall x_1: t_1^o \ldots \forall x_n: t_n^o[f])$. We only treat the case $f \equiv \lambda \eta : (k_1, \ldots , k_m)$; the other cases are similar. $z \in FV(f)$, say: $z \equiv x_p$. As $x_p: t_p^o \in \Gamma$, $a_p = a - 1$. Hence $\text{ord}_\Gamma (x) = \text{ord}_\Gamma (T(x^o)) = 1 = a - 1$. By 2.29, $\text{ord}_\Gamma (zT(k_1) \ldots T(k_m)) \leq a - 1$. Hence

$$\text{ord}_\Gamma (T(z_1: t_1^o, \ldots , z_n: t_n^o[f])) = \text{ord}_\Gamma (\Pi z_1: T(t_1^o) \ldots \Pi z_n: T(t_n^o). zT(k_1) \ldots T(k_m))$$

$$= \max(\text{ord}_\Gamma (zT(k_1) \ldots T(k_m)), \text{ord}_\Gamma (T(t_1^o)))$$

$$= \max(\text{ord}_\Gamma (T(z_1^o, \ldots , z_n^o)), \max(a, 1)) = a + 1 = a$$

□

Corollary 4.7 If $\Gamma \vdash_R f : (\forall x: c^0)$ then $\text{ord}_\Gamma (T(f)) = a$.

5 Conclusions

In this paper we gave a description in PTS format of a fragment of the type theory behind the proof checker Nuprl. We showed that the obtained system has some basic properties of PTSs (Church Rosser, Generation Lemma, Correctness of Types, Subject Reduction, and a weak, but not restrictive, form of Unicity of Types).

Our formulation of Nuprl as a PTS is faithful to the idea behind universes in Martin-Löf's type theory as in [15, 16], where universes are built à la Russell in that $\Pi$, $\Sigma$, etc. are both set (or type) forming operations and operations to form canonical elements of universes. This is clear in our compact formulation of the Nuprl fragment where Martin-Löf's universes $U_i$ are $\epsilon_i$ for us, Martin-Löf's II-formation, II-introduction and II-elimination correspond to our (II-form), (A) and (App) respectively. Of course in PTS style, there are no equality rules, but (Conv) plays some role of these equality rules. As for $\Sigma$, which we replaced by $\times$, it is obvious that (Pairs), (Left) and (Right) play the role of $\llcorner \Sigma \lrcorner$-introduction and elimination.

Most important of all, is our definition of the concept of order on Nuprl terms which captures the notion of Type Universe of Martin-Löf's formulation à la Russell and is related to Type Universe

---

5In [17] Martin-Löf built universes à la Tarski where a distinction was made between sets and their codes.
in Nuprl [9] (and to Degree of Expression in AUTOMATH [18]). Orders in Nuprl are also shown to be related to orders in the Ramified Type Theory RTT of Principia Mathematica [24] in a precise way: If we translate a proposition \( f \) of order \( n \) in RTT into a Nuprl type \( T(f) \) by means of a propositions-as-types embedding, the order of \( T(f) \) in Nuprl is also \( n \).

As a bonus, the embedding \( T \) results in a description of RTT in a propositions-as-types style in which the notion of order is maintained. Moreover, a PTS description of RTT is obtained.

There are more similarities between RTT and Nuprl. Both Nuprl and RTT have a kind of higher order substitution (see Chapter 5 of [10] and Section 3 of [14]). We are currently investigating the similarities between both notions of substitution.

References

<table>
<thead>
<tr>
<th>Number</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>93/01</td>
<td>R. van Geldrop</td>
<td>Deriving the Aho-Corasick Algorithm: A Case Study into the Synergy of Programming Methods, p. 36.</td>
</tr>
<tr>
<td>93/02</td>
<td>T. Verhoeff</td>
<td>A Continuous Version of the Prisoner's Dilemma, p. 17</td>
</tr>
<tr>
<td>93/03</td>
<td>T. Verhoeff</td>
<td>Quicksort for Linked Lists, p. 8</td>
</tr>
<tr>
<td>93/04</td>
<td>E.H.L. Aarts, J.H.M. Korst, P.J. Zwietering</td>
<td>Deterministic and Randomized Local Search, p. 78</td>
</tr>
<tr>
<td>93/05</td>
<td>J.C.M. Baeten, C. Verhoef</td>
<td>A Congruence Theorem for Structured Operational Semantics with Predicates, p. 18</td>
</tr>
<tr>
<td>93/06</td>
<td>J.P. Veltkamp</td>
<td>On the Unavoidability of Metastable Behaviour, p. 29</td>
</tr>
<tr>
<td>93/07</td>
<td>P.D. Moerland</td>
<td>Exercises in Multiprogramming, p. 97</td>
</tr>
<tr>
<td>93/08</td>
<td>J. Verhoosel</td>
<td>A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32</td>
</tr>
<tr>
<td>93/10</td>
<td>K.M. van Hee</td>
<td>Systems Engineering: A Formal Approach Part II: Frameworks, p. 44</td>
</tr>
<tr>
<td>93/12</td>
<td>K.M. van Hee</td>
<td>Systems Engineering: A Formal Approach Part IV: Analysis Methods, p. 63</td>
</tr>
<tr>
<td>93/14</td>
<td>J.C.M. Baeten, J.A. Bergstra</td>
<td>On Sequential Composition, Action Prefixes and Process Prefix, p. 21</td>
</tr>
<tr>
<td>93/15</td>
<td>J.C.M. Baeten, J.A. Bergstra, R.N. Bof</td>
<td>A Real-Time Process Logic, p. 31</td>
</tr>
<tr>
<td>93/16</td>
<td>H. Schepers, J. Hooman</td>
<td>A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems, p. 27</td>
</tr>
<tr>
<td>93/17</td>
<td>D. Alstein, P. van der Stok</td>
<td>Hard Real-Time Reliable Multicast in the DEDOS System, p. 19</td>
</tr>
<tr>
<td>93/18</td>
<td>C. Verhoeff</td>
<td>A Congruence Theorem for Structured Operational Semantics with Predicates and Negative Premises, p. 22</td>
</tr>
<tr>
<td>93/19</td>
<td>G.J. Houben</td>
<td>The Design of an Online Help Facility for ExSpect, p. 21</td>
</tr>
<tr>
<td>93/20</td>
<td>F.S. de Boer</td>
<td>A Process Algebra of Concurrent Constraint Programming, p. 15</td>
</tr>
<tr>
<td>93/21</td>
<td>M. Codish, D. Danis, G. Filé, M. Bruynooghe</td>
<td>Freeness Analysis for Logic Programs - And Correctness, p. 24</td>
</tr>
<tr>
<td>93/22</td>
<td>E. Poll</td>
<td>A Typechecker for Bijective Pure Type Systems, p. 28</td>
</tr>
<tr>
<td>93/23</td>
<td>E. de Kogel</td>
<td>Relational Algebra and Equational Proofs, p. 23</td>
</tr>
<tr>
<td>93/24</td>
<td>E. Poll and Paula Severi</td>
<td>Pure Type Systems with Definitions, p. 38</td>
</tr>
<tr>
<td>93/26</td>
<td>W.M.P. van der Aalst</td>
<td>Multi-dimensional Petri Nets, p. 25</td>
</tr>
<tr>
<td>93/27</td>
<td>T. Kloks and D. Kratsch</td>
<td>Finding All Minimal Separators of a Graph, p. 11</td>
</tr>
<tr>
<td>93/28</td>
<td>F. Kamareddine and R. Nederpelt</td>
<td>A Semantics for a Fine λ-Calculus with de Bruijn Indices, p. 49</td>
</tr>
<tr>
<td>93/29</td>
<td>R. Post and P. De Brabandere</td>
<td>GOLD, a Graph-Oriented Language for Databases, p. 42</td>
</tr>
<tr>
<td>93/30</td>
<td>J. Deogun, T. Kloks, D. Kratsch, H. Müller</td>
<td>On Vertex Ranking for Permutation and Other Graphs, p. 11</td>
</tr>
</tbody>
</table>
Derivation of delay insensitive and speed independent CMOS circuits, using directed commands and production rule sets, p. 40.


ILIAS, a sequential language for parallel matrix computations, p. 20.

Real Time Process Algebra with Infinitesimals, p. 39.

Abstract Reduction and Topology, p. 28.

Non Interleaving Process Algebra, p. 17.

Design and Analysis of Dynamic Leader Election Protocols in Broadcast Networks, p. 73.

A general conservative extension theorem in process algebra, p. 17.

Job Shop Scheduling by Constraint Satisfaction, p. 22.


Temporal operators viewed as predicate transformers, p. 11.

Automatic Verification of Regular Protocols in P/T Nets, p. 23.

A taxonomy of finite automata construction algorithms, p. 87.

A taxonomy of finite automata minimization algorithms, p. 23.

A precise clock synchronization protocol, p.


Verifying Sequentially Consistent Memory using Interface Refinement, p. 20.

The object-oriented paradigm, p. 28.

Canonical typing and II-conversion, p. 51.


Graph Isomorphism Models for Non Interleaving Process Algebra, p. 18.


Time and the Order of Abstract Events in Distributed Computations, p. 29.


A Hierarchical Diagrammatic Representation of Class Structure, p. 22.

Process Algebra with Partial Choice, p. 16.
The testing Paradigm Applied to Network Structure. p. 31.


A New Method for Integrity Constraint checking in Deductive Databases. p. 34.

Ups and Downs of Type Theory. p. 9.

Job Shop Scheduling by Local Search. p. 21.

Mathematical Induction Made Calculational. p. 36.

An Algebraic Semantics of Basic Message Sequence Charts. p. 9.

Refining Reduction in the Lambda Calculus. p. 15.

The performance of single-keyword and multiple-keyword pattern matching algorithms. p. 46.

Beyond β-Reduction in Church's λ→. p. 22.


The design and implementation of the FIRE engine: A C++ toolkit for Finite automata and regular Expressions.

An algebraic semantics of Message Sequence Charts. p. 43.

Abstract Interpretation of Reactive Systems: Abstractions Preserving VCTL*, 3CTL* and CTL*, p. 28.

K_{1,3}-free and W_{4}-free graphs. p. 10.

On the foundations of functional programming: a programmer's point of view. p. 54.


Correctness of Real Time Systems by Construction. p. 22.

Process Algebra with Feedback. p. 22.

A Boyer-Moore type algorithm for regular expression pattern matching. p. 22.


A formalization of the Ramified Type Theory, p.40.

The Barendregt Cube with Definitions and Generalised Reduction, p. 37.

Delayed choice: an operator for joining Message Sequence Charts, p. 15.

Canonical typing and Π-conversion in the Barendregt Cube, p. 19.

Simulating and Analyzing Railway Interlockings in ExSpect, p. 30.

Point-free substitution. p. 10.
On the equivalence covering number of splitgraphs, p. 4.
Distributed Consensus and Hard Real-Time Systems, p. 34.
Computing a perfect edge without vertex elimination ordering of a chordal bipartite graph, p. 6.
Concatenation of Graphs, p. 7.
Category Theory as Coherently Constructive Lattice Theory: An Illustration, p. 35.
Verifying Sequentially Consistent Memory, p. 160
Tutorial voor de ExSpect-bibliotheek voor "Administratieve Logistiek", p. 43.
The A-cube with classes of terms modulo conversion, p. 16.
On Π-conversion in Type Theory, p. 12.
Fixed-Point Calculus, p. 11.
Process Algebra with Propositional Signals, p. 25.
A short and flexible proof of Strong Normalization for the Calculus of Constructions, p. 27.
Listing simplicial vertices and recognizing diamond-free graphs, p. 4.
Traces and Logic, p. 81
A Partial Order Approach to Branching Time Logic Model Checking, p. 20.
The Construction of a small Communication Library, p. 16.
Formalizing Process Algebraic Verifications in the Calculus of Constructions, p. 49.
Concrete process algebra, p. 134.
An Isotopic Invariant for Planar Drawings of Connected Planar Graphs, p. 9.
A Type Inference Algorithm for Pure Type Systems, p. 20.
A Quantitative Analysis of Iterated Local Search, p. 23.
Drawing Execution Graphs by Parsing, p. 10.
Preservation of Strong Normalisation for Explicit Substitution, p. 12.
Discrete Time Process Algebra, p. 20
MathPad: A System for On-Line Preparation of Mathematical Documents, p. 15
Deductive Database Systems and integrity constraint checking, p. 36.

Empty Interworkings and Refinement Semantics of Interworkings Revised, p. 19.


De proceedings: ACP'95, p.

On the Connection of Partial Order Logics and Partial Order Reduction Methods, p. 12.

Abstract Interpretation of Reactive Systems: Preservation of CTL*, p. 27.

Specification of tools for Message Sequence Charts, p. 36.

A Reflection on Russell’s Ramified Types and Kripke’s Hierarchy of Truths, p. 14.

Discrete Time Process Algebra with Abstraction, p. 15.

On Normalisation, p. 33.

Axiomatizing Early and Late Input by Variable Elimination, p. 44.


Petri net based scheduling, p. 20.


Practical Symbolic Model Checking of the full $\mu$-calculus using Compositional Abstractions, p. 17.


Context-Free Graph Grammars and Concatenation of Graphs, p. 35.

Record concatenation with intersection types, p. 46.

An algebraic semantics for hierarchical P/T Nets, p. 32.

Process Algebra with Autonomous Actions, p. 12.


Example specifications in phi-SDL.


Life-Cycle Inheritance A Petri-Net-Based Approach, p. 18.

Structural Petri Net Equivalence, p. 16.


AUTOMATH and Pure Type Systems, p. 30.