Faithful Reproductions of the Automath Landau Formalization

Chad E. Brown

Saarland University, Saarbrücken, Germany

Abstract. In the 1970s van Benthem Jutting translated Landau's book *Grundlagen der Analysis* into Automath. The type theory of Automath differs from modern type theories in significant ways. The most important difference is that Automath is λ-typed while most modern type theories are Π-typed. In this paper we define the notion of a faithful reproduction of an Automath signature. We describe how one can obtain a particular faithful reproduction of a signature corresponding to the Automath version of Landau's book. We then consider what properties a Π-typed type theory must have in order for the faithful reproduction to be a correct signature. Based on this information, we can give different mappings of the reproduction into type theories such as ECC and, in particular, the system Coq.

Keywords: Automath, Extended Calculus of Constructions, Type Theory, Proof Assistants, Proof Checking

1 Introduction

The first large scale mathematical formalization was the Automath version of Landau's book *Grundlagen der Analysis* [9] carried out by van Benthem Jutting [7]. There are some aspects of the Automath language that have not been incorporated into modern type theories. First and foremost, Automath is λ-typed while most modern type theories are Π-typed. (An exception is $\lambda\delta$ [6], which is a modern λ-typed type theory. We should note that the van Benthem Jutting formalization has been ported to $\lambda\delta$ [5].) Another unusual aspect of the Automath language AUT-QE, in which the Landau formalization was carried out, is so-called type inclusion. Type inclusion allows one to use, e.g., predicates as propositions.

There is an obvious mapping from the syntax of a Π-typed type theory to the syntax of a λ-typed type theory: simply change every Π to be a λ. This mapping is far from injective, of course. Given a correct signature in a λ-typed type theory, one may ask whether there is a preimage of this mapping that is also a correct signature in a Π-typed type theory. We explore this question with some particular signatures corresponding to the van Benthem Jutting Automath book. It will turn out that one can find an appropriate preimage, assuming one does a few η-expansions. The fact that these η-expansions are necessary is due to type inclusion in AUT-QE.

Once we have an appropriate preimage, we can ask what properties the Π-typed type theory must have so that the preimage is a correct signature. A contextual variant of LF would be enough, but a traditional version of LF is not. The Extended Calculus of Constructions (ECC) is also enough. Finally, we discuss a particular port of the appropriate preimage to the proof assistant Coq.[1]

2 Syntax and Type Theories

We start by giving enough syntax to describe both the Automath type theory AUT-QE and the Π-typed type theories we will consider. Let $\mathcal{V}$ be a countably infinite set of variables, $\mathcal{C}$ be a countably infinite set of constants and $\mathcal{S}$ be a set of sorts. Let `prop` and `type` be distinct sorts in $\mathcal{S}$. These are the only two sorts in AUT-QE. We use x, y, z to range over variables, c, d to range over constants and s to range over sorts. We define terms, substitutions, contexts and signatures recursively. We use M, N, A, B to range over terms, θ, φ to range over substitutions, Γ to range over contexts and Σ to range over signatures.

$$\text{Terms } (M, N, A, B) ::= x|s|[\theta]c|(\Pi x : A.M)|(\lambda x : A.M)|MN$$

$$\text{Substitutions } (\theta, \varphi) ::= \cdot|\theta, x := M$$

$$\text{Contexts } (\Gamma) ::= \odot|\Gamma, x : A$$

$$\text{Signatures } (\Sigma) ::= \bullet|\Sigma, c : (\Gamma \rhd A)|\Sigma, c : (\Gamma \rhd M : A)$$

We write nonempty contexts as $x_1 : A_1, \cdots, x_n : A_n$ instead of $\odot, x_1 : A_1, \cdots, x_n : A_n$. We adopt similar conventions for nonempty substitutions and nonempty signatures. We also assume the usual notion of a variable x being free in a term M. We say x is free in a substitution θ if there is some $y := M$ in θ where x is free in M. We consider α-equivalent terms to be the same. We assume no variable is listed twice in the domain of a substitution or context. Likewise, we assume no constant is listed twice in the domain of a signature.

It should be noted that we are diverging from the customary notation for Automath. For example, an application MN is written as $< N > M$ in Automath notation. Also, in Automath each occurrence of a constant c is accompanied with a list of parameters $M_1, \ldots, M_n$ written as $c(M_1, \ldots, M_n)$. The corresponding notation in our case is $[\theta]c$ where θ is the substitution $x_1 := M_1, \ldots, x_n := M_n$, where the variables x_i are determined by the context of the declaration of c.

There are substitution operations θM and $\theta \circ \varphi$ allowing us to apply a substitution θ to a term M or substitution φ defined as follows.

- $\theta x := M$ if $x := M$ is in θ. Otherwise, $\theta x := x$.
- $\theta s := s$
- $\theta[\varphi]c := [\theta \circ \varphi]c$

[1] Files corresponding to the results reported in this paper are available at `http://www.ps.uni-saarland.de/∼cebrown/landau/index.php`

- $\theta(\mathit{\Pi} x : A.M) := (\mathit{\Pi} y : \theta A.(\theta, x := y)M)$ where y is chosen, depending on θ, x, A and M, to avoid capture.
- $\theta(\lambda x : A.M) := (\lambda y : \theta A.(\theta, x := y)M)$ where y is chosen to avoid capture.
- $\theta(MN) := (\theta M)(\theta N)$
- $\theta \circ \cdot := \cdot$
- $\theta \circ (\varphi, x := M) := (\theta \circ \varphi), x := \theta M$

We define β and η redexes and their reducts as usual:

$$(\lambda x : A.M)N \text{ is a } \beta\text{-redex with reduct } [x := N]M$$

$$(\lambda x : A.Mx) \text{ is an } \eta\text{-redex with reduct } M \text{ if } x \text{ is not free in } M$$

Conversely, if x is not free in M, we say $\lambda x : A.Mx$ is an η-expansion of M. We can define three congruent reflexive transitive notions of reduction in the usual way. We write $M \rightarrow_\beta N$ to mean M β-reduces to N in 0 or more steps (where the reduction may occur at any subterm position). We use $M \rightarrow_\eta N$ and $M \rightarrow_{\beta\eta} N$ to mean the corresponding notion for η or both β and η. For each $* \in \{\beta, \eta, \beta\eta\}$ we write $M \equiv_* N$ for the corresponding equivalence relations. We can also apply the equivalence relation $\equiv_*$ to compare two substitutions, two contexts or two signatures in an obvious way.

Let $\mathit{\Sigma}$ be a signature. Relative to $\mathit{\Sigma}$, we have a notion of δ-reduction.

$$[\theta]c \text{ is a } \delta\text{-redex with reduct } \theta M \text{ if } c : (\mathit{\Gamma} \rhd M : A) \text{ is in } \mathit{\Sigma}$$

We write $\rightarrow^{\mathit{\Sigma}}_{\delta\beta}$ for the congruent reflexive transitive relation induced by δ-reduction and β-reduction, and $\equiv^{\mathit{\Sigma}}_{\delta\beta}$ for the corresponding equivalence relation. We can also apply these relations to substitutions and contexts. We will not apply these relations to signatures, since they are relative to a signature already.

There are several type theories in the Automath family. The only one of interest in this paper is AUT-QE, as this is the type theory in which Landau's *Grundlagen* was formalized. The definition of AUT-QE can be found in [4]. For the purpose of this paper, we will not need the full definition of AUT-QE. The first important fact is that no $\mathit{\Pi}$-abstraction $\mathit{\Pi} x : A.M$ will appear in any Automath signature, context, substitution or term. All the other language constructions will be used. In Section 3 we will describe Automath signatures corresponding to the van Benthem Jutting formalization of Landau's book. There we will discuss particular aspects of the AUT-QE type theory that are relevant.

We now turn to a $\mathit{\Pi}$-typed type theory. We cannot directly use a pure type system since we wish to discuss signatures. Furthermore, we wish to include contextual declarations in signatures as Automath makes extensive use of them. In most implemented systems, declarations are global, i.e., are not relative to a context $\mathit{\Gamma}$. This is true in the Twelf system [13]. In terms of our notation, a declaration in a Twelf signature $\mathit{\Sigma}$ must be of the form $c : (\odot \rhd A)$ or $c : (\odot \rhd M : A)$. In recent years, work has been done on contextual type theories [11, 1]. While the aim of the work on contextual type theories is the treatment of meta-variables, this could also give a way to treat declarations in context. A practical way to make some contextual definitions in Twelf could be to make use of the recent module system [14]. (This abuse of the module system should work unless there is an occurrence of a term $[\theta]c$ where a variable free in θ is bound in the context in which it occurs. There are such occurrences of terms in the Automath *Grundlagen* book.) In systems such as Coq [3], definitions can be made within a context of variables, but they become global definitions in the signature. This is possible because in type theories such as ECC (Extended Calculus of Constructions) [10] there are enough universes to guarantee that definitions in context can always be replaced by global definitions where application is used instead of substitution. That is, we could use $[\cdot]c\,N_1 \cdots N_n$ instead of $[x_1 := N_1, \ldots, x_n := N_n]c$. Barendregt gives a pure type system that corresponds (more or less) to AUT-QE (see page 216 of [2]). A "parking place" sort Δ is used to allow for "certain terms." In practice these "certain terms" would correspond to the subterms of the form $[\cdot]cN_1 \cdots N_k$ with $k < n$. This mechanism is described in more detail (in the context of AUT-68) in [8].[2] A compromise position is taken in the $\lambda 68$ system of [8]. In $\lambda 68$, definitions are made global and application is used instead of substitution, but the definitions are made global using new binders § (instead of λ) and ¶ (instead of Π). These new binders prevent terms like $\S x : A.B$ from being "first class citizens" of the type theory.

We will follow the tradition of pure type systems by defining a type theory $\mathcal{T}$ relative to a set $\mathcal{A}$ of axioms and a set $\mathcal{R}$ of rules, but depart from the tradition of pure type systems in two other ways. First, we must include signatures with contextual declarations. We support contextual declarations by including substitutions and always pairing constants with substitutions. Second, we define the type theory in an algorithmic style.[3]

We consider the set $\mathcal{S}$ of sorts to be fixed, with `prop` and `type` playing a special role. Let us further suppose there are two other distinct sorts `kindp` and `kind`. An *axiom* is a pair $s_1 : s_2$ of sorts. Axioms are used to indicate that a sort is a member of another sort. We will commonly assume axioms $\texttt{prop} : \texttt{kindp}$ and $\texttt{type} : \texttt{kind}$. A *rule* is a triple (s_1, s_2, s_3) of sorts. A rule is used to indicate that a sort is closed under certain Π-abstractions. We follow the usual convention (see [2]) of writing (s_1, s_2) for the rule (s_1, s_2, s_2).

Given sets $\mathcal{A}$ of axioms and $\mathcal{R}$ of rules, the *type theory* $\mathcal{T} := \mathcal{T}(\mathcal{A}, \mathcal{R})$ will consist of four judgments:

- $\vdash_{\mathcal{T}} \Sigma\,\texttt{sig}$ meaning Σ is a correct signature.
- $\Sigma \vdash_{\mathcal{T}} \Gamma\,\texttt{ctx}$ meaning that Γ is a correct context, if Σ is a correct signature.
- $\Sigma; \Gamma \vdash_{\mathcal{T}} M : A$ meaning that M has type A, if Σ and Γ are correct.
- $\Sigma; \Gamma \vdash_{\mathcal{T}} \theta : \Gamma'$ meaning θ substitutes the variables in $\texttt{dom}(\Gamma')$ within the context Γ, assuming Σ and Γ are correct.

[2] According to Section 3.1 of [8], the idea to use such a sort Δ for this purpose originated with van Benthem Jutting.

[3] The rules give an abstract description of a simple checker the author has implemented. This checker has been used to justify the claims about which rules are sufficient to check signatures corresponding to the *Grundlagen*.

$$\textit{Axiom}\quad \frac{(s_1, s_2) \in \mathcal{A}}{\Sigma; \Gamma \vdash s_1 : s_2} \qquad\qquad \textit{Var}\quad \frac{x : A \in \Gamma}{\Sigma; \Gamma \vdash x : A}$$

$$\textit{Prim}\quad \frac{c : (\Gamma' \vdash A) \in \Sigma \qquad \Sigma; \Gamma \vdash \theta : \Gamma'}{\Sigma; \Gamma \vdash [\theta]c : \theta A}$$

$$\textit{Def}\quad \frac{c : (\Gamma' \vdash M : A) \in \Sigma \qquad \Sigma; \Gamma \vdash \theta : \Gamma'}{\Sigma; \Gamma \vdash [\theta]c : \theta A}$$

$$\textit{Ap}\quad \frac{\Sigma; \Gamma \vdash M : (\Pi x : A.B) \qquad \Sigma; \Gamma \vdash N : A}{\Sigma; \Gamma \vdash MN : ([x := N]B)} \qquad \lambda\quad \frac{\Sigma; \Gamma, x : A \vdash M : B}{\Sigma; \Gamma \vdash (\lambda x : A.M) : (\Pi x : A.M)}$$

$$\Pi\quad \frac{\Sigma; \Gamma \vdash A : s_1 \qquad \Sigma; \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathcal{R}}{\Sigma; \Gamma \vdash (\Pi x : A.B) : s_3}$$

$$\textit{Conv}\quad \frac{\Sigma; \Gamma \vdash M : A \qquad A \equiv^{\Sigma}_{\delta\beta} B}{\Sigma; \Gamma \vdash M : B} \qquad\qquad \textit{SubstE}\quad \frac{}{\Sigma; \Gamma \vdash \cdot : \odot}$$

$$\textit{SubstC}\quad \frac{\Sigma; \Gamma \vdash \theta : \Gamma' \qquad \Sigma; \Gamma \vdash M : \theta A}{\Sigma; \Gamma \vdash \theta, x := M : \Gamma', x : A} \qquad\qquad \textit{CtxE}\quad \frac{}{\Sigma \vdash \odot\, \mathtt{ctx}}$$

$$\textit{CtxC}\quad \frac{\Sigma \vdash \cdot : \Gamma\, \mathtt{ctx} \qquad \Sigma; \Gamma \vdash A : s}{\Sigma \vdash \Gamma, x : A\, \mathtt{ctx}} \qquad\qquad \textit{SigE}\quad \frac{}{\vdash \Sigma\, \mathtt{sig}}$$

$$\textit{SigPrim}\quad \frac{\vdash \Sigma\, \mathtt{sig} \qquad \Sigma \vdash \Gamma\, \mathtt{ctx} \qquad \Sigma; \Gamma \vdash A : s}{\vdash \Sigma, (c : \Gamma \rhd A)\, \mathtt{sig}}$$

$$\textit{SigDef}\quad \frac{\vdash \Sigma\, \mathtt{sig} \qquad \Sigma \vdash \Gamma\, \mathtt{ctx} \qquad \Sigma; \Gamma \vdash A : s \qquad \Sigma; \Gamma \vdash M : A}{\vdash \Sigma, c : (\Gamma \rhd M : A)\, \mathtt{sig}}$$

Fig. 1. Type theory $\mathcal{T}(\mathcal{A}, \mathcal{R})$

These judgments are defined by the rules in Figure 1. Note that there is no cumulativity of universes as in ECC, and there is no use of η-conversion. We use $\delta\beta$-conversion (relative to the current signature) to check definitional equality. We make no major claims about the meta-theory of such a type theory $\mathcal{T}$, and do not intend this paper to be a study of such type theories.

In order to define the notion of a faithful reproduction of an Automath signature, we define an operator that erases the distinction between Π-binders and λ-binders by mapping both to λ-binders. We define $\lfloor - \rfloor$ on all terms, substitutions, contexts and signatures as follows:

$$\lfloor x \rfloor := x \qquad \lfloor s \rfloor := s \qquad \lfloor ([\theta]c) \rfloor := [\lfloor \theta \rfloor]\lfloor c \rfloor \qquad \lfloor (\Pi x : A.M) \rfloor := \lambda x : \lfloor A \rfloor.\lfloor M \rfloor$$

$$\lfloor (\lambda x : A.M) \rfloor := \lambda x : \lfloor A \rfloor.\lfloor M \rfloor \qquad \lfloor (MN) \rfloor := \lfloor M \rfloor \lfloor N \rfloor \qquad \lfloor \cdot \rfloor := \cdot$$

$$\lfloor (\theta, x := M) \rfloor := (\lfloor \theta \rfloor, x := \lfloor M \rfloor) \qquad \lfloor \odot \rfloor := \odot \qquad \lfloor \Gamma, x : A \rfloor := \lfloor \Gamma \rfloor, x : \lfloor A \rfloor$$

$$\lfloor \bullet \rfloor := \bullet \qquad \lfloor (\Sigma, c : (\Gamma \vdash A)) \rfloor := \lfloor \Sigma \rfloor, c : (\lfloor \Gamma \rfloor \vdash \lfloor A \rfloor)$$

$$\lfloor (\Sigma, c : (\Gamma \vdash M : A)) \rfloor := \lfloor \Sigma \rfloor, c : (\lfloor \Gamma \rfloor \vdash \lfloor M \rfloor : \lfloor A \rfloor)$$

Let Σ be a signature and $\mathcal{T}$ be a type theory. A *faithful reproduction of Σ in $\mathcal{T}$* is a signature Σ' such that $\vdash_{\mathcal{T}} \Sigma' \,\texttt{sig}$ and $\lfloor \Sigma' \rfloor$ is the same as Σ. For each $* \in \{\beta, \eta, \beta\eta\}$, a *faithful reproduction of Σ in $\mathcal{T}$ modulo* $*$ is a signature Σ' such that $\vdash_{\mathcal{T}} \Sigma' \,\texttt{sig}$ and $\lfloor \Sigma' \rfloor \equiv_* \Sigma$.

3 The Automath Landau Book

Information about the Automath formalization of Landau can be found in [7, 15]. The Automath "book" is a file with 10702 lines. This book determines a signature we will denote by $\Sigma_{\mathbf{A}}$. The signature declares 32 primitive (undefined) constants and 6878 defined constants, for a total of 6910 signature elements. Each of these signature elements is declared on a unique line. The remaining lines either open or close "paragraphs" or change the current context Γ. The paragraph system of Automath was important part of making the formalization process realistic since it allowed for names to be reused without conflicts. We can safely ignore the paragraph system when discussing the signature $\Sigma_{\mathbf{A}}$. To avoid conflicts with names, as well as to help locate signature elements in the original Automath book, we incorporate the Automath paragraph names as prefixes to the original signature element name. For example, one of the constants declared in $\Sigma_{\mathbf{A}}$ is `l_e_is`. In the Automath book, the corresponding name is simply `is`, which is declared in the paragraph `e` which is a subparagraph of the (global) paragraph `l`. When no conflict will arise, we will omit unnecessary prefixes for the sake of readability.

We describe a few of the first declarations in $\Sigma_{\mathbf{A}}$, but only those we will need in the subsequent discussion.

- `imp` : $(a : \texttt{prop}, b : \texttt{prop} \rhd (\lambda x : a.b) : \texttt{prop})$ defines implication using λ in a way that is quite unique to Automath. In a Π-typed type theory, implication would be represented by $\Pi x : a.b$. We will write $M \Rightarrow N$ for the term $[a := M, b := N]\texttt{imp}$.
- `con` : $(\odot \rhd \texttt{prop})$ assumes `con` to play the role of a false proposition. It is in the empty context. We will write $\bot$ to mean the term $[\cdot]\texttt{con}$.
- `not` : $(a : \texttt{prop} \rhd (a \Rightarrow \bot) : \texttt{prop})$ defines negation. We will write $\neg M$ for the term $[a := M]\texttt{not}$.
- `wel` : $(a : \texttt{prop} \rhd \neg(\neg a) : \texttt{prop})$ defines `wel` to be the double negation operation.

- $\mathtt{et} : (a : \mathtt{prop}, w : ([a := a]\mathtt{wel}) \rhd a)$ declares $\mathtt{et}$ as the axiom that makes the logic classical by allowing one to prove a proposition a by proving its double negation. We will write $\mathtt{et}(M, N)$ for the term $[a := M, w := N]\mathtt{et}$.
- $\mathtt{all} : (\sigma : \mathtt{type}, p : (\lambda x : \sigma.\mathtt{prop}) \rhd p : \mathtt{prop})$ is intended to represent the universal quantifier over a type. We will write $\forall_M N$ as notation for the term $[\sigma := M, p := N]\mathtt{all}$.

The definition of $\mathtt{all}$ already means there will not be a faithful reproduction of $\Sigma_{\mathbf{A}}$ in our Π-typed type theories. The reproduction of $\lambda x : \sigma.\mathtt{prop}$ must have a sort as a type, and so it must be $\Pi x : \sigma.\mathtt{prop}$. Assuming $\Pi x : \sigma.\mathtt{prop} \not\equiv^{\Sigma}_{\delta\beta} \mathtt{prop}$, one can easily see that $\Sigma; \sigma : \mathtt{type}, p : \Pi x : \sigma.\mathtt{prop} \not\vdash_{\mathcal{T}} p : \mathtt{prop}$ by examining the rules in Figure 1.[4]

The only reason the definition of $\mathtt{all}$ is correct in AUT-QE is that AUT-QE has type inclusion [4]. In general, type inclusion means any term of the form $\lambda x_1 : M_1 \cdots \lambda x_n : M_n \lambda y : N.s$ also has type $\lambda x_1 : M_1 \cdots \lambda x_n : M_n.s$ (where s is either $\mathtt{type}$ or $\mathtt{prop}$). In this particular case, p has both type $\lambda x : \sigma.\mathtt{prop}$ and (by type inclusion) type $\mathtt{prop}$ in AUT-QE.

If we η-expand the definition of $\mathtt{all}$ to be $\lambda x : \sigma.px$, then in order to obtain a faithful reproduction we will only need to have the following:

$$\Sigma; \sigma : \mathtt{type}, p : \Pi x : \sigma.\mathtt{prop} \vdash_{\mathcal{T}} \Pi x : \sigma.px : \mathtt{prop}.$$

There is also another reason to η-expand the definition of $\mathtt{all}$. As is discussed in Section 4.1.1 of [7], η-reduction is only used twice to check that $\Sigma_{\mathbf{A}}$ is a correct AUT-QE signature. Both of these cases can be avoided by changing the definitions of two signature elements. The first is to change the definition of $\mathtt{all}$ to be $\lambda x : \sigma.px$. The second is to change the definition of $\mathtt{r\_imp}$, which is a dependent form of implication. The declaration of $\mathtt{r\_imp}$ in $\Sigma_{\mathbf{A}}$ is as follows:

- $\mathtt{r\_imp} : (a : \mathtt{prop}, b : \lambda x : a.\mathtt{prop} \rhd b : \mathtt{prop})$

These two definitions can be easily changed in the Automath book, and Wiedijk's Automath checker [15] verifies that η-reduction is no longer needed to check the book. Let $\Sigma^1_{\mathbf{A}}$ be the signature that corresponds to this modified book. That is, $\Sigma^1_{\mathbf{A}}$ is the same signature as $\Sigma_{\mathbf{A}}$ with the two exceptions that the definitions of $\mathtt{all}$ and $\mathtt{r\_imp}$ in $\Sigma^1_{\mathbf{A}}$ are $\lambda x : \sigma.px$ and $\lambda x : a.bx$, respectively.

We now turn to the signature element $\mathtt{non}$.

$$\mathtt{non} : (\sigma : \mathtt{type}, p : (\lambda x : \sigma.\mathtt{prop}) \rhd (\lambda x : \sigma.\neg(px)) : \lambda x : \sigma.\mathtt{prop})$$

We will write $\mathtt{non}(M, N)$ for the term $[\sigma := M, p := N]\mathtt{non}$. The definition of $\mathtt{non}$ could be reproduced in a type theory $\mathcal{T}$ in two different ways: $\Pi x : \sigma.\neg(px)$ or $\lambda x : \sigma.\neg(px)$. The first option is the proposition meaning the predicate p is empty. The second option defines the complement of the predicate p. We will further discuss this ambiguity below.

[4] Since $\lambda x : \sigma.\mathtt{prop}$ and $\mathtt{prop}$ are $\delta\beta$-normal, $\Pi x : \sigma.\mathtt{prop} \not\equiv^{\Sigma}_{\delta\beta} \mathtt{prop}$ follows from confluence of $\rightarrow^{\Sigma}_{\delta\beta}$.

We now consider the signature element some_t1, which corresponds to a simple lemma. The context in which some_t1 is defined consists of five variable declarations: $\sigma : \texttt{type}$, $p : (\lambda x : \sigma.\texttt{prop})$, $n : \neg(\forall_\sigma p)$, $m : \texttt{non}(\sigma, \texttt{non}(\sigma, p))$, $s : \sigma$. The type is ps, and the definition (proof term) is $\texttt{et}(ps, ms)$. Note that neither the definition nor the type depend on the variable n. Also, none of the types of the variables that follow n in the context depend on n. In such a case we could clearly simplify the declaration of some_t1 to omit n from the context.

The fact that some definitions have such unnecessary dependencies is an artifact of the Automath language. In the Automath language it is very easy to declare a variable in context and make use of it in several declarations that follow. If the variable is in context when a definition is made, it will remain part of the context of the definition even if it is not used. This also means that every occurrence of the defined constant will be associated with a substitution that substitutes for the irrelevant variable.

These irrelevant context variables are uninteresting but make the task of finding a faithful reproduction more difficult. For this reason, we computed a simplified signature $\Sigma^2_{\mathbf{A}}$ from $\Sigma^1_{\mathbf{A}}$ in which the contexts of all definitions are reduced to the smallest correct context in which the type and definition make sense. (The contexts of primitive constants were not changed.) It turned out that 24% (8843 out of 36837) of the variables in contexts could be removed. (The signature $\Sigma^2_{\mathbf{A}}$ has been checked to be a correct AUT-QE signature by Wiedijk's Automath checker [15].)

Except some_t1, the declarations of the constants discussed above are the same in $\Sigma^2_{\mathbf{A}}$ and $\Sigma^1_{\mathbf{A}}$. The context of some_t1 in $\Sigma^2_{\mathbf{A}}$ is

$$\sigma : \texttt{type},\ p : (\lambda x : \sigma.\texttt{prop}),\ m : \texttt{non}(\sigma, \texttt{non}(\sigma, p)),\ s : \sigma$$

Before we leave some_t1, we discuss the type $\texttt{non}(\sigma, \texttt{non}(\sigma, p))$ given for the variable m. In AUT-QE, the subterm $\texttt{non}(\sigma, p)$ has type $\lambda x : \sigma.\texttt{prop}$. By type inclusion, the term $\texttt{non}(\sigma, p)$ also has type prop. We must consider the subterm $\texttt{non}(\sigma, p)$ as having type $\lambda x : \sigma.\texttt{prop}$ in order to use it as it is in $\texttt{non}(\sigma, \texttt{non}(\sigma, p))$. Intuitively, we think of $\texttt{non}(\sigma, p)$ as being the complement of the predicate p. Likewise, $\texttt{non}(\sigma, \texttt{non}(\sigma, p))$ has type $\lambda x : \sigma.\texttt{prop}$ and prop. In this case, to use the term as a type for m, we must consider it has having the type prop. Intuitively, $\texttt{non}(\sigma, \texttt{non}(\sigma, p))$ means the complement of p is empty.

Here we see a fundamental difficulty in obtaining faithful reproductions of Automath signatures. It is not simply the case that some λ-binders correspond to λ's while other correspond to Π's. The λ in the definition of non is used in both ways even within a single term. (The point that it is not clear whether a λ in AUT-QE should be a λ or a Π is discussed in Section 5.2 of [8]. This is in contrast to AUT-68, for which a translation to a Π-typed type theory is given in Definition 23 of [8].) If we will be satisfied with a faithful reproduction modulo η, then there is a simple solution in this case. We treat λ as a λ in the definition of non, but wrap the occurrence of non with a Π when required. In this particular case, we can use $\Pi x : \sigma.\texttt{non}(\sigma, \texttt{non}(\sigma, p))x$ as the type of m in the context of the definition of some_t1. Note that applying $\lfloor - \rfloor$ to this term yields $\lambda x : \sigma.\texttt{non}(\sigma, \texttt{non}(\sigma, p))x$ which η-reduces to the desired Automath term.

4 A Faithful Reproduction

There is a signature Σ_Π such that $\lfloor \Sigma_\Pi \rfloor \equiv_\eta \Sigma^2_{\mathbf{A}}$. This signature was mostly obtained using an ad hoc algorithm to recognize when some λs should be changed to Πs. Roughly speaking, using Automath terminology, λ-abstractions of degree 1 are changed to Π-abstractions, λ-abstractions of degree 3 are left alone, and to decide whether to change a λ-abstraction of degree 2 to a Π-abstraction, analyze how it is used. We dynamically η-expanded and later η-reduced again after deciding which λs should be Πs. There is a type theory $\mathcal{T}$ for which $\vdash_{\mathcal{T}} \Sigma_\Pi\ \mathtt{sig}$ holds. For any such $\mathcal{T}$, Σ_Π is a faithful reproduction of $\Sigma^2_{\mathbf{A}}$ modulo η.

We conjecture that there is an appropriate type theory $\mathcal{T}$ (any of the ones discussed in Section 5) and an algorithm which can always compute a faithful reproduction in $\mathcal{T}$ modulo η of a correct AUT-QE signature. In practical terms, this would mean any correct AUT-QE signature could be ported to modern systems such as Coq.

A natural question to ask is whether $\lfloor \Sigma_\Pi \rfloor$ is itself a correct AUT-QE signature. It turns out that it is. Also, the Automath checker confirms that η-conversion is not needed to check $\lfloor \Sigma_\Pi \rfloor$.

Let $\Sigma^3_{\mathbf{A}}$ be $\lfloor \Sigma_\Pi \rfloor$. The signature $\Sigma^3_{\mathbf{A}}$ differs from $\Sigma^2_{\mathbf{A}}$ in that some subterms of 25 declarations in $\Sigma^3_{\mathbf{A}}$ must be η-reduced to obtain $\Sigma^2_{\mathbf{A}}$. Consequently, $\Sigma^3_{\mathbf{A}}$ only differs from the original signature $\Sigma_{\mathbf{A}}$ by doing some η-expansions and by removing unnecessary variables from the contexts of declarations. Hence $\Sigma^3_{\mathbf{A}}$ is arguably the same as the signature one obtains from the van Benthem Jutting Automath version of Landau's book. Also, Σ_Π is clearly a faithful reproduction of $\Sigma^3_{\mathbf{A}}$ in any type theory $\mathcal{T}$ for which $\vdash_{\mathcal{T}} \Sigma_\Pi\ \mathtt{sig}$.

For comparison, in Figure 2 we show some of the signature elements discussed in Section 3 in both $\Sigma^2_{\mathbf{A}}$ and Σ_Π. The type of m in the context of `some_t1` shows the only example in Figure 2 where an η-expansion was required to introduce a λ that is reproduced as a Π. Of course, if we had started from $\Sigma_{\mathbf{A}}$, then we would have needed to η-expand the definitions of `all` and `r_imp` as well.

Almost all of the η-expansions are with occurrences of `non`. The only examples of η-expansions that are not associated with a `non` occur in `r_ande2`. `r_ande2` corresponds to a natural deduction rule for a dependent conjunction. In `r_ande2`, several occurrences of a variable b of type $\Pi x : a.\mathtt{prop}$ (where $a : \mathtt{prop}$) are wrapped in a Π to form a term of type `prop`. The context of `r_ande2` is of the form $\ldots, a_1 : [a := a, b := b]\mathtt{and}$ in $\Sigma^2_{\mathbf{A}}$. After η-expanding an occurrence of b and changing the λ to a Π, the context of `r_and` has the form $\ldots, a_1 : [a := a, b := (\Pi x : a.bx)]\mathtt{and}$ in Σ_Π. The same process is repeated with one occurrence of b in the type of `r_ande2` and two occurrences of b in the definition of `r_ande2`. Note that all of these η-expansions are required to account for the fact that AUT-QE has type inclusion, but $\mathcal{T}$ does not.

We also note that all of the η-expansions are either required in the logical preliminaries of the Automath book (before the content of Landau begins) or in Chapter 1 of Landau's book. To be more specific, the only η-expansions that are required in the part of the Automath book corresponding to Landau's book

Element	Context	Type	Definition
`imp` ($\Sigma^2_{\mathbf{A}}$)	$a : \texttt{prop}$, $b : \texttt{prop}$	$\texttt{prop}$	$(\lambda x : a.b)$
(Σ_Π)	$a : \texttt{prop}$, $b : \texttt{prop}$	$\texttt{prop}$	$(\Pi x : a.b)$
`all` ($\Sigma^2_{\mathbf{A}}$)	$\sigma : \texttt{type}$, $p : (\lambda x : \sigma.\texttt{prop})$	$\texttt{prop}$	$(\lambda x : \sigma.px)$
(Σ_Π)	$\sigma : \texttt{type}$, $p : (\Pi x : \sigma.\texttt{prop})$	$\texttt{prop}$	$(\Pi x : \sigma.px)$
`non` ($\Sigma^2_{\mathbf{A}}$)	$\sigma : \texttt{type}$, $p : (\lambda x : \sigma.\texttt{prop})$	$(\lambda x : \sigma.\texttt{prop})$	$(\lambda x : \sigma.(\neg px))$
(Σ_Π)	$\sigma : \texttt{type}$, $p : (\Pi x : \sigma.\texttt{prop})$	$(\Pi x : \sigma.\texttt{prop})$	$(\lambda x : \sigma.(\neg px))$
`some_t1` ($\Sigma^2_{\mathbf{A}}$)	$\sigma : \texttt{type}$, $p : (\lambda x : \sigma.\texttt{prop})$, $m : \texttt{non}(\sigma, \texttt{non}(\sigma, p))$, $s : \sigma$	ps	$\texttt{et}(ps, ms)$
(Σ_Π)	$\sigma : \texttt{type}$, $p : (\Pi x : \sigma.\texttt{prop})$, $m : (\Pi x : \sigma.\texttt{non}(\sigma, \texttt{non}(\sigma, p))x)$, $s : \sigma$	ps	$\texttt{et}(ps, ms)$
`r_imp` ($\Sigma^2_{\mathbf{A}}$)	$a : \texttt{prop}$, $b : (\lambda x : a.\texttt{prop})$	$\texttt{prop}$	$(\lambda x : a.bx)$
(Σ_Π)	$a : \texttt{prop}$, $b : (\Pi x : a.\texttt{prop})$	$\texttt{prop}$	$(\Pi x : a.bx)$

Fig. 2. Some signature elements in $\Sigma^2_{\mathbf{A}}$ and Σ_Π

are in some lemmas used to prove *Satz 27*. *Satz 27* is a theorem stating that the natural numbers are well-ordered.

5 Type Theories for the Reproduction

We have already fixed four distinct sorts `prop`, `type`, `kindp` and `kind`. We fix the set $\mathcal{A}_0$ of axioms to be $\texttt{prop} : \texttt{kindp}$ and $\texttt{type} : \texttt{kind}$. We could identify `kindp` and `kind`, but for now it is worth pointing out that they need not be identified. We fix the set $\mathcal{R}_0$ of 6 rules as follows:

$$\begin{aligned}\mathcal{R}_0 := \{\ &(\texttt{prop}, \texttt{prop}), (\texttt{type}, \texttt{prop}), (\texttt{type}, \texttt{type}),\\ &(\texttt{prop}, \texttt{type}), (\texttt{type}, \texttt{kindp}), (\texttt{prop}, \texttt{kindp})\ \}\end{aligned}$$

Let $\mathcal{T}_0$ be $\mathcal{T}(\mathcal{A}_0, \mathcal{R}_0)$. This type theory $\mathcal{T}_0$ is sufficient to obtain $\vdash_{\mathcal{T}_0} \Sigma_\Pi\ \texttt{sig}$.

Consideration of four signature elements is sufficient to justify the inclusion of each of the rules in $\mathcal{R}_0$. Two of these elements have already been discussed: `all` and `r_imp`. Their contexts, types and definitions can be found in Figure 2. (In the judgments below, we omit the signature. The intended signature is the part of Σ_Π preceding the declaration of the signature element in question.) To check the definition of `all`, we must check

$$\sigma : \texttt{type},\ p : (\Pi x : \sigma.\texttt{prop}) \vdash_{\mathcal{T}_0} \Pi x : \sigma.px : \texttt{prop}$$

which requires the rule $(\texttt{type}, \texttt{prop})$. In order for the context of `all` to be correct, we must have $\sigma : \texttt{type} \vdash_{\mathcal{T}_0} (\Pi x : \sigma.\texttt{prop}) : s$ for some sort s. We could introduce a new sort s and include the rule $(\texttt{type}, \texttt{kindp}, s)$ to handle this case. We follow the simpler alternative of using `kindp` as the sort s and including the rule $(\texttt{type}, \texttt{kindp})$. Similarly, to check the correctness of the context of `r_imp`, $a : \texttt{prop}$, $b : (\Pi x : a.\texttt{prop})$, we must have $a : \texttt{prop} \vdash_{\mathcal{T}_0} (\Pi x : a.\texttt{prop}) : s$ for some sort s. We include the rule ($\texttt{prop}, \texttt{kindp}$) to handle this case. The definition and type of $\texttt{r\_imp}$ require

$$\texttt{r\_imp} : a : \texttt{prop}, b : \lambda x : a.\texttt{prop} \vdash_{\mathcal{T}_0} \Pi x : b.bx : \texttt{prop}$$

which makes the inclusion of the rule ($\texttt{prop}, \texttt{prop}$) mandatory.

Without the rule ($\texttt{type}, \texttt{type}$) we could not even form simple types. Still, it is worth examining a declaration in Σ_Π to justify the inclusion of this rule. Consider the primitive declaration $\texttt{fisi}$ corresponding to the axiom of functional extensionality. The declaration of $\texttt{fisi}$ in Σ_Π makes use of the primitive element $\texttt{is}$ representing (polymorphic) book equality:

– $\texttt{is} : (\sigma : \texttt{type}, s : \sigma, t : \sigma \rhd \texttt{prop})$. We use $M =_A N$ as notation for the term $[\sigma := A, s := M, t := N]\texttt{is}$.

Let Γ denote the context of $\texttt{fisi}$:

$$\sigma : \texttt{type},\ \tau : \texttt{type},\ f : (\Pi x : \sigma.\tau),\ g : (\Pi x : \sigma.\tau),\ i : (\Pi x : \sigma.(fx =_\tau gx))$$

The type of $\texttt{fisi}$ is $(f\ =_{(\Pi x:\sigma.\tau)}\ g)$. In order for the signature Σ_Π to be correct, we need $\Gamma \vdash_{\mathcal{T}_0} (f\ =_{(\Pi x:\sigma.\tau)}\ g) : s$ for some sort s. Given the type of σ in the context of $\texttt{is}$, we need $\Gamma \vdash_{\mathcal{T}_0} (\Pi x : \sigma.\tau) : \texttt{type}$, justifying the rule ($\texttt{type}, \texttt{type}$).

Finally, we must justify the inclusion of the rule ($\texttt{prop}, \texttt{type}$). For this reason we consider the signature element $\texttt{seq}$, an element makes use of three other definitions in Σ_Π. It is enough to consider the contexts and types of these three elements, so we omit the definitions.

– $\texttt{real} : (\odot \rhd \cdots : \texttt{type})$ is the type of reals. We write $\texttt{real}$ for $[\cdot]\texttt{real}$.
– $\texttt{intrl} : (r : \texttt{real} \rhd \cdots : \texttt{prop})$ is the predicate that recognizes when a real is an integer. We write $\texttt{intrl}(M)$ for the term $[r := M]\texttt{intrl}$.
– $\texttt{lessis} : (r : \texttt{real},\ s : \texttt{real} \rhd \cdots : \texttt{prop})$ is the relation between reals r and s that is true when $r \leq s$. We write $M \leq N$ for $[r := M, s := N]\texttt{lessis}$.

Let Γ be $x : \texttt{real}$, $y : \texttt{real}$, $\alpha : \texttt{type}$, the context of $\texttt{seq}$. The type of $\texttt{seq}$ is $\texttt{type}$. The definition of $\texttt{seq}$ is

$$\Pi t : \texttt{real}.\Pi u : \texttt{intrl}(t).\Pi v : (y \leq t).\Pi w : (t \leq x).\alpha.$$

Let x and y be real numbers and α be a type. If there are n integers in the interval $[x, y]$, then $\texttt{seq}$ corresponds to a type of n-tuples of elements of type α. A set-theoretic way to interpret the definition is as the function space

$$\{t \in [x, y] | t \text{ is an integer}\} \to \alpha.$$

Clearly we need

$$\Gamma \vdash_{\mathcal{T}_0} (\Pi t : \texttt{real}.\Pi u : \texttt{intrl}(t).\Pi v : (y \leq t).\Pi w : (t \leq x).\alpha) : \texttt{type}.$$

The simplest way to ensure this is to include the rule ($\texttt{prop}, \texttt{type}$). Note that this rule is stronger than what is required. By including the rule ($\texttt{prop}, \texttt{type}$), we have $p : \texttt{prop}, \alpha : \texttt{type} \vdash_{\mathcal{T}_0} (\Pi v : p.\alpha) : \texttt{type}$, which is not necessary to have $\vdash_{\mathcal{T}_0} \Sigma_\Pi$. It would be enough to include a new sort `ctype` (for "conditional" types) and the rules $(\texttt{prop}, \texttt{type}, \texttt{ctype})$, $(\texttt{prop}, \texttt{ctype})$ and $(\texttt{type}, \texttt{ctype}, \texttt{type})$.

If we wished to work in a type theory without contextual declarations, then we could use global definitions and replace substitution with application. Without going into technical details, let $\Sigma_\Pi^\odot$ be a signature corresponding to Σ_Π in the following sense (using M' to refer to the term that corresponds to each M):

1. Every declaration of the form
$$c : (x_1 : N_1, \ldots, x_n : N_n \rhd A) \text{ or } c : (x_1 : N_1, \ldots, x_n : N_n \rhd M : A)$$
in Σ_Π is of the form
$$c : (\odot \rhd \forall x_1 : N_1' \cdots \forall x_n : N_n'.A')$$
$$\text{or } c : (\odot \rhd \lambda x_1 : N_1' \cdots \lambda x_n : N_n'.M' : \forall x_1 : N_1' \cdots \forall x_n : N_n'.A')$$
(respectively) in $\Sigma_\Pi^\odot$.
2. Every occurrence of a constant $[M_1, \ldots, M_n]c$ in Σ_Π is replaced by $[\cdot]c\, M_1' \cdots M_n'$ in $\Sigma_\Pi^\odot$.[5]

We do not have $\vdash_{\mathcal{T}_0} \Sigma_\Pi^\odot\, \texttt{sig}$; we need more rules. The type of `all` in $\Sigma_\Pi^\odot$ is

$$\Pi\sigma : \texttt{type}.\Pi p : (\Pi x : \sigma.\texttt{prop}).\texttt{prop}.$$

We know $\sigma : \texttt{type} \vdash_{\mathcal{T}_0} (\Pi x : \sigma.\texttt{prop}) : \texttt{kindp}$ and $\sigma : \texttt{type}, p : (\Pi x : \sigma.\texttt{prop}) \vdash_{\mathcal{T}_0} \texttt{prop} : \texttt{kindp}$. There is no rule $(\texttt{kindp}, \texttt{kindp}, s)$ in $\mathcal{R}_0$ for any sort s. We could consider using `kindp` for s and adding the rule $(\texttt{kindp}, \texttt{kindp})$. However, if we add the rule $(\texttt{kindp}, \texttt{kindp})$, then we could not only form $(\Pi p : (\Pi x : \sigma.\texttt{prop}).\texttt{prop})$ but also higher types such as $(\Pi q : (\Pi p : (\Pi x : \sigma.\texttt{prop}).\texttt{prop}).\texttt{prop})$. To avoid this extra strength, we can use a "parking place" sort Δ.

Once we have fixed the new sort Δ, we add 13 new rules to the rules in $\mathcal{R}_0$, all of which allow us to form new Π-abstractions in Δ. We define $\mathcal{R}_1$ as follows:

$$\begin{aligned}\mathcal{R}_1 := \mathcal{R}_0 \cup \{\ & (\texttt{type}, \texttt{kind}, \Delta), (\texttt{type}, \Delta), (\texttt{kind}, \texttt{kind}, \Delta), \\ & (\texttt{kindp}, \texttt{kind}, \Delta), (\texttt{kind}, \texttt{type}, \Delta), (\texttt{kindp}, \texttt{type}, \Delta), \\ & (\texttt{kind}, \texttt{prop}, \Delta), (\texttt{kind}, \texttt{kindp}, \Delta), (\texttt{prop}, \Delta), \\ & (\texttt{kind}, \Delta), (\texttt{kindp}, \texttt{prop}, \Delta), (\texttt{kindp}, \Delta), (\texttt{kindp}, \texttt{kindp}, \Delta)\ \}\end{aligned}$$

We have $\vdash_{\mathcal{T}_1} \Sigma_\Pi^\odot\, \texttt{sig}$ for $\mathcal{T}_1 := \mathcal{T}(\mathcal{A}_0, \mathcal{R}_1)$.

Suppose we choose to identify the sorts `kindp` and `kind` (and call them both $\square$). Since `kind` does not appear in any of the rules of $\mathcal{T}_0$, this would have no significant effect on this theory. On the other hand, 6 of the additional 13 rules of $\mathcal{T}_1$ would be duplicates. It turns out, however, that in order to check $\Sigma_\Pi^\odot$ the rule $(\texttt{type}, \square, \Delta)$ is no longer needed if `kindp` and `kind` are the same. Let $\mathcal{A}_2$ consist of $\texttt{prop} : \square$ and $\texttt{type} : \square$ and $\mathcal{R}_2$ consist of the following 12 rules:

$$\begin{gathered}(\texttt{prop}, \square), (\texttt{type}, \square), (\texttt{type}, \texttt{type}), (\texttt{prop}, \texttt{type}), (\texttt{type}, \texttt{prop}), (\texttt{prop}, \texttt{prop}), \\ (\square, \texttt{prop}, \Delta), (\square, \texttt{type}, \Delta), (\square, \square, \Delta), (\texttt{type}, \Delta), (\texttt{prop}, \Delta), (\square, \Delta)\end{gathered}$$

[5] This is similar to the treatment of constants depending on parameters in Definition 23 of [8].

We define $\mathcal{T}_2$ to be $\mathcal{T}(\mathcal{A}_2, \mathcal{R}_2)$ and note that $\vdash_{\mathcal{T}_2} \Sigma_\Pi^{\odot}\,\texttt{sig}$.

In addition to identifying the sorts `kindp` and `kind`, we could, of course, identify `prop` and `type`. This would have some unfortunate consequences. In particular, the constant `et` is declared in Σ_Π as a primitive notion with context $a : \texttt{prop}$, $w : [a := a]\texttt{wel}$ and type a. This is intended to allow one to prove propositions by contradiction. If `prop` is the same as `type`, then we could apply `et` to types as well as propositions. Essentially, `et` would act as a choice operator. (This point is discussed in Section 4.1.2 of [7].)

Nevertheless, suppose we do identify `prop` and `type` (and call them $*$) and identify `kindp` and `kind` (and call them $\Box$). In this case, $\mathcal{R}_0$ reduces to be simply

$$\{(*, \Box), (*, *)\}$$

which corresponds to a contextual version of the type theory LF. This also corresponds to the extension of $\lambda 68$ corresponding to AUT-QE given in Section 5.2 of [8].

In order to check $\Sigma_\Pi^{\odot}$, it is enough to have a type theory $\mathcal{T}_3 := \mathcal{T}(\mathcal{A}_3, \mathcal{R}_3)$ where $\mathcal{A}_3$ is $\{* : \Box\}$ and $\mathcal{R}_3$ consists of the 6 rules

$$(*, \Box), (*, *), (\Box, *, \Delta), (\Box, \Box, \Delta), (*, \Delta), (\Box, \Delta)$$

This is the pure type system corresponding to AUT-QE given on page 216 of [2].

6 Mapping the Reproduction into ECC and Coq

In general, a translation of Σ_Π or $\Sigma_\Pi^{\odot}$ into another type theory will be determined by the mapping of the sorts `type` and `prop`. Assuming the rules of Figure 1 are admissible in the target theory, we can verify the signature maps to a correct target signature by also giving sorts corresponding to `kind`, `kindp` and Δ and checking that the axioms and rules of one of the type theories of Section 5 hold in the target theory.

Coq is a proof assistant [3] based on the Calculus of Inductive Constructions. As mentioned before, while declarations in Coq may be made within a context of variables, they are always made global and given a global type. Hence we only discuss translating $\Sigma_\Pi^{\odot}$ into Coq. For the purpose of type checking $\Sigma_\Pi^{\odot}$, we only need the ECC fragment [10]. The sorts (or "universes") of ECC consist of an impredicative universe **Prop** and a hierarchy of predicative universes $\mathbf{Type}_n$ for $n \geq 0$. These universes are cumulative: **Prop** is a subuniverse of each $\mathbf{Type}_n$ and $\mathbf{Type}_n$ is a subuniverse of $\mathbf{Type}_m$ whenever $n \leq m$.

The translation into Coq will be determined by the translation of `type` and `prop`. While the most obvious translation mapping `prop` to **Prop** and `type` to $\mathbf{Type}_0$ suffices, it is worth noting that other translations will also map into a correct ECC signature. Suppose we decide to map `type` and `prop` to ECC universes U_t and U_p, respectively. By cumulativity, there is some natural number k such that $U_t : \mathbf{Type}_k$ and $U_p : \mathbf{Type}_k$. For this reason, it is enough to consider the type theory $\mathcal{T}_2$ in which `kind` and `kindp` are identified as $\Box$. We can translate

$\square$ and Δ to be $\mathbf{Type}_k$. By our choice of k, the axioms `type` : $\square$ and `prop` : $\square$ will map to axioms that are satisfied in ECC. The 12 rules of $\mathcal{T}_2$ map to 11 rules:

$$(U_p, \mathbf{Type}_k), (U_t, \mathbf{Type}_k), (U_t, U_t), (U_p, U_t), (U_t, U_p), (U_p, U_p), (U_t, \mathbf{Type}_k),$$
$$(\mathbf{Type}_k, U_p, \mathbf{Type}_k), (\mathbf{Type}_k, U_t, \mathbf{Type}_k), (\mathbf{Type}_k, \mathbf{Type}_k), (U_p, \mathbf{Type}_k)$$

Many of the rules correspond to rules that are either clearly in ECC, or are in ECC by the choice of k. The only rules that may not be in ECC are (U_p, U_t) and (U_t, U_p). There are essentially three correct ways to choose U_p and U_t.

1. Choose U_p to be **Prop** and U_t to be $\mathbf{Type}_n$ (where k will be chosen such that $k > n$). Here we know $(\mathbf{Prop}, \mathbf{Type}_n)$ is a rule of ECC since **Prop** is a subuniverse of $\mathbf{Type}_n$. We know $(\mathbf{Type}_n, \mathbf{Prop})$ is a rule of ECC since **Prop** is impredicative.
2. We may reverse the roles of the sorts, choosing U_t to be **Prop** and U_p to be $\mathbf{Type}_n$ (where k will be chosen such that $k > n$).
3. We may choose U_p and U_t to be the same universe (either **Prop** or $\mathbf{Type}_n$ for some n).

In the system Coq, one simply writes **Type** for $\mathbf{Type}_n$, leaving the system to check for consistency of the implicit universe indices. Each of the three options above gives a correct Coq file. Using the Coq universe **Set**, one can find even more possibilities for translating into Coq. Some of these options are only correct if one chooses **Set** to be impredicative.

In spite of these choices, there is a motivation for choosing the "obvious" option of sending `type` to **Type** and `prop` to **Prop**. Recall that 32 of the declarations in the Automath book are primitive notions. We can define most of these primitive notions in Coq if we map `type` to **Type** and `prop` to **Prop**.

The last primitive notions declared in the Automath book consist of the type `nat` and the Peano axioms. These are the only primitive notions corresponding to the Landau book. The other primitive notions are in the logical preliminaries. If we map `type` to **Type**, then we can use Coq's inductive types to define `nat` and prove the Peano axioms. (We can even do this if we map both `type` and `prop` to **Type**. The induction axiom in this case will be a recursion operator.) If we map `type` to **Prop**, we can also define an inductive proposition `nat`, but the Peano axioms will not be provable. In fact, if `nat` is a **Prop**, then its members are proofs. The Peano axioms imply there is more than one `nat`, hence more than one proof of `nat`. In Coq, one can prove proof irrelevance from excluded middle.[6] Consequently, if we map `type` to **Prop** in Coq, then the primitive notions of $\Sigma_{\Pi}^{\odot}$ will lead to an inconsistency.

Five of the primitive notions in the Automath book correspond to assuming that for each type σ there is a type of sets over σ. This essentially makes the underlying logic higher-order. Consider the corresponding primitive notions in $\Sigma_{\Pi}^{\odot}$ (omitting the empty contexts and empty substitutions):

– `set` : $\Pi\sigma$: `type.type` corresponds to a type of sets over the type σ.

[6] This is true for the Calculus of Inductive Constructions, an extension of ECC.

- `esti` : $\Pi\sigma$: `type`.$\Pi s : \sigma.\Pi s_0$: (`set` σ).`prop` is a membership relation between an element s and a set s_0.
- `setof` : $\Pi\sigma$: `type`.Πp : ($\Pi x : \sigma$.`prop`).(`set` σ) maps each predicate p over σ to a set over σ.
- `estii` and `estie` assert equivalence of (`esti` $\sigma\, s$ (`setof` $\sigma\, p$)) and ps.

Since ECC is a higher-order type theory, there is an easy way to appropriately define these in Coq: simply use the function type $\sigma \to$ **Prop** as the type of sets over σ. Application can be used to define `esti`, and `setof` can be defined by simply returning p. The remaining definitions are trivial. in Coq. If we choose to both map `type` and `prop` to **Type**, then the definition of `setof` would be ill-typed, as p would have type $\sigma \to$ **Type** and we would need to return something of type $\sigma \to$ **Prop**. Of course, we could consider changing the definition of `set` to be $\lambda\sigma$: `type`.$\sigma \to$ **Type**. This choice leads to a universe inconsistency in Coq later when set types are used to define quotient types.

Suppose we map `type` to **Type** and `prop` to **Prop**. We can define 26 of the 32 primitive notions in Coq. We briefly consider the 6 remaining primitive notions which cannot be defined in Coq. They correspond to axioms that are commonly assumed when Coq is being used to formalize classical mathematics.

- `et` is the formulation of the double negation law considered earlier. If we define `con` to be **False** in Coq, `et` will be precisely the double negation law.
- `ind` corresponds to an indefinite description operator and `oneax` is the axiom that `ind` behaves as an indefinite description operator.
- `fisi` is an axiom of functional extensionality.
- `otax1` can be described as follows. The signature element `ot` allows one to construct a new type `ot`(σ, p) from a given type σ and a predicate p over σ. The signature element `in` is a function from `ot`(σ, p) to σ. Now, `otax1` is the axiom stating that `in` is injective. A natural way to define `ot`(σ, p) in Coq is using a Σ-type (an inductive type) of pairs (x, u) where $x : \sigma$ and $u : px$. Of course, `in` can then be defined as the first projection sending (x, u) to x. We can only prove `otax1` is injective if we have proof irrelevance.
- `isseti` is an axiom of set extensionality. Assuming we define sets as predicates in Coq, `isseti` will correspond to extensionality of predicates.

We conclude with a brief remark about Martin-Löf style type theories with predicative universes but no impredicative universe. The analysis above clearly demonstrates that one can map $\Sigma_{\Pi}^{\odot}$ into such a type theory by mapping `type` and `prop` to the same universe. As noted earlier, identifying `type` and `prop` means that the double negation law will give a choice principle.

7 Conclusion

We have defined a notion of a faithful reproduction of an Automath signature in a Π-typed type theory. After omitting redundant context variables and performing a few η-expansions to handle type inclusion, the signature of the Automath

Landau formalization can be faithfully reproduced in a type theory such as ECC (in different ways). We have described one rendering of such a reproduction as a Coq file. In this Coq version, all the primitive notions of the Automath book can be defined in the Coq version, if one assumes classical logic, extensionality principles, and a description operator.

References

1. Andreas Abel and Brigitte Pientka. Explicit Substitutions for Contextual Type Theory. In Karl Crary and Marino Miculan, editors, *LFMTP*, volume 34 of *EPTCS*, pages 5–20, 2010.
2. Henk Barendregt. Lambda calculi with types. In S. Abramsky, Dov M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, pages 117–309. Clarendon Press, 1992.
3. Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.
4. D.T. van Daalen. A description of Automath and some aspects of its language theory. In P. Braffort, editor, *Proceedings of the Symposium APLASM*, volume I, 1973. Also in [12].
5. F. Guidi. Landau's "Grundlagen der Analysis" from Automath to lambda-delta. Technical Report UBLCS 2009-16, University of Bologna, Bologna, Italy, September 2009.
6. F. Guidi. The Formal System $\lambda\delta$. *Transactions on Computational Logic*, 11(1):Article No. 5, October 2009.
7. L.S. van Benthem Jutting. *Checking Landau's "Grundlagen" in the AUTOMATH System.* PhD thesis, Eindhoven University of Technology, 1977.
8. Fairouz Kamareddine, Twan Laan, and Rob Nederpelt. Automath and pure type systems. *Electr. Notes Theor. Comput. Sci.*, 85(7), 2003.
9. E. Landau. *Grundlagen der Analysis.* Leizig, 1930.
10. Zhaohui Luo. *Computation and reasoning: a type theory for computer science.* Oxford University Press, Inc., New York, NY, USA, 1994.
11. Aleksandar Nanevski, Frank Pfenning, and Brigitte Pientka. Contextual modal type theory. *ACM Trans. Comput. Logic*, 9:23:1–23:49, June 2008.
12. R.P. Nederpelt, J.H. Geuvers, and R.C. de Vrijer, editors. *Selected Papers on Automath.* Studies in Logic and the Foundations of Mathematics **133**. North-Holland, Amsterdam, 1994.
13. Frank Pfenning and Carsten Schurmann. System Description: Twelf — A Meta-Logical Framework for Deductive Systems. In *Proceedings of the 16th International Conference on Automated Deduction (CADE-16*, pages 202–206. Springer-Verlag LNAI, 1999.
14. Florian Rabe and Carsten Schürmann. A practical module system for LF. In *Proceedings of the Fourth International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice*, LFMTP '09, pages 40–48, New York, NY, USA, 2009. ACM.
15. Freek Wiedijk. A New Implementation of Automath. *J. Autom. Reasoning*, 29(3-4):365–387, 2002.