Verified Representations of Landau's "Grundlagen" in the $\lambda\delta$ Family and in the Calculus of Constructions

Ferruccio Guidi
Department of Computer Science and Engineering
University of Bologna
Mura Anteo Zamboni 7, 40127, Bologna, ITALY
e-mail: ferruccio.guidi@unibo.it

Landau's "Grundlagen der Analysis" formalized in the language Aut-QE, represents an early milestone in computer-checked mathematics and is the only non-trivial development finalized in the languages of the Automath family. Here we discuss an implemented procedure producing a faithful representation of the Grundlagen in the Calculus of Constructions, verified by the proof assistant Coq 8.4.3. The point at issue is distinguishing λ-abstractions from Π-abstractions where the original text uses Automath unified binders, taking care of the cases in which a binder corresponds to both abstractions at one time. It is a fact that some binders can be disambiguated only by verifying the Grundlagen in a calculus accepting Aut-QE and the Calculus of Constructions. To this end, we rely on $\lambda\delta$ version 3, a system that the author is proposing here for the first time.

Our representation of the Grundlagen as a user-level script that Coq 8.4.3 accepts without warnings, is hosted on $\lambda\delta$ Web site at <http://lambdadelta.info/download/grundlagen_2.v>.

1. INTRODUCTION

Jutting's representation [vB79] of Landau's "*Grundlagen der Analysis*" [Lan65] in the Automath language Aut-QE [vD94a] is an early milestone in computer-checked mathematics and is the only non-trivial development finalized in the languages of the Automath family [NGdV94]. Actually, the development of significant formalized mathematics on the grounds of the Grundlagen is limited in that the current proof-checking software based on Aut-QE [Wie02] seems incapable to compete with the most recent proof management systems. Thus, some authors propose translations of Aut-QE into Pure Type Systems [Bar93, KLN03], which give the background for making the Grundlagen accessible to systems like Coq [Coq15], which accepts the Calculus of Constructions [CH88] (henceforth, CC for short).

Here we discuss a farther step: an implemented procedure producing a representation of the Grundlagen in CC. This representation has the form of a user-level script that is available at <http://lambdadelta.info/download/grundlagen_2.v> and that Coq 8.4.3 can process without reporting errors or warnings.

To our knowledge, the only related work is [Bro11], where the author reports on representations of the Grundlagen in several Contextual Pure Type Systems [AP10] originating from ECC [Luo90]. One of these representations yields a user-level script for Coq, that is very close to ours. Nevertheless, we claim that our script is a closer approximation of the original Automath text, that we produce with a simpler and more innovative translation procedure from Aut-QE.

Firstly, we recall some concepts about CC and Aut-QE for convenience.

The terms of both systems are organized in three classes of increasing degree (kinds, types, and elements) comprising two sorts denoting the universes of sets and propositions, references by name, applications, and binding abstractions.

On the one hand, CC has distinct abstractions $\lambda_x N.M$ and $\Pi_x N.M$ for functions and function spaces respectively. On the other hand, Aut-QE is best known for its unified abstraction `[x:N]M` representing both abstractions of CC, as well as for its reversed notational convention for application with respect to CC. In particular, we remind the reader that the application $(M\ N)$ in CC appears as `<N>M` in Aut-QE.

The latter convention, that is assumed also in the calculus we shall present in Section 3.1, helps the visual understanding of redexes as [KN96] explains.

Being an Automath text, the formalized Grundlagen amounts to a list of (almost 7000) constants declared or defined within a system of sections (known as paragraphs), and introduced in a context of unified binders (known as block openers).

Considering the verification of an Automath text from the CC perspective, unified abstraction yields no logical confusion as long as the verifier can separate λ-abstractions and Π-abstractions during type conversion and type inference.

On the contrary, when the author of the text assumes the unpleasant equality $\lambda_x N.M = \Pi_x N.M$, which in Aut-QE is an identity, and which implies $\Pi_x N.S = S$ if S is the sort typing M, then logical confusion is unavoidable since, for instance, a predicate is equated to the formula representing its universal quantification.

Unfortunately, this inconvenience affects the Grundlagen indeed, as we see in the case of the constant `all"l"`, whose name stands for "$\forall$-introduction", defined as the function $\lambda_\sigma \mathrm{Type}.\lambda_p(\Pi_x \sigma.\mathrm{Prop}).p$ that maps the predicate p to itself, instead of mapping it to the formula $\Pi_x \sigma.(p\ x)$ that is the universal quantification of p.

We want to stress that the Π-introduction $p \mapsto \Pi_x \sigma.(p\ x)$ reads in Aut-QE as `p` $\mapsto$ `[x:sigma]<x>p`, exactly like the η-reduction $p \mapsto \lambda_x \sigma.(p\ x)$. For this reason some authors address these transformations with misleading terminology.

interestingly, the mechanical translation of the Grundlagen into CC is not an issue once references are resolved, and unified binders are disambiguated.

While static analysis suffices to resolve all references in the Grundlagen, and to disambiguate the majority of its approximately 47000 unified binders, almost 3000 such binders can be disambiguated only by observing their reductional behavior during verification in a calculus that accepts Aut-QE and CC at once.

To this end, we rely on $\lambda\delta$ version 3, a system that the author is proposing in this article for the first time. Other calculi of the same family appear in [Gui09b, Gui15].

In particular, the mechanical translation reported in this article works as follows: by static analysis (Section 2), followed by dynamic analysis (Section 3), we build a representation of the Grundlagen in $\lambda\delta$ version 3. This is mapped straightforwardly to a slightly refined CC (Section 4), and coded for Coq 8.4.3. Our conclusions are in Section 5, where we show some benchmarks and some examples of the translation.

We find it convenient to agree on the next terminology used henceforth:

—the QE-GdA will refer to Landau's "*Grundlagen*" presented in Aut-QE;

—the $\lambda\delta$-GdA will refer to the QE-GdA presented in $\lambda\delta$ version 3; a *raw* version will have ambiguous binders, while a *proper* version will have disambiguated binders;

—the CC-GdA will refer to the proper $\lambda\delta$-GdA presented in the refined CC.

name:	`x`	::= identifier	Barendegt's convention is assumed
term:	`M,N`	::= `'type','prop'`	sorts
		| `x`	reference to name `x`
		| `[x:N]M`	typed abstraction of `x` in `M`
		| `<N>M`	application of `M` to `N`
context:	`C`	::= `@`	empty context
		| `[x:N]C`	local typed declaration of `x` before `C`
book:	`B`	::= `;`	empty book
		| `C x:=M:N B`	global typed definition of `x` in `C` before `B`
		| `C x:='prim':N B`	global typed declaration of `x` in `C` before `B`

Fig. 1. The abstract syntax of an Automath language.

2. STATIC ANALYSIS OF THE "GRUNDLAGEN"

Landau's "Grundlagen der Analysis" [Lan65] contains 301 propositions on the arithmetics of rational, irrational and complex numbers. This theory was digitally specified in the language Aut-QE [vD94a] by Jutting [vB77]. Later, it was recovered from Jutting's original files by Wiedijk, who included it in the latest distribution of his validator for Aut-68 [vB94a] and Aut-QE.

Unfortunately, we have scarce practical information on the specification.

Here is a summary of what we know up to now [Gui09a].

—The *concrete syntax*, found in [Wie99], (see Section 2.1) relies on the next facilities meant to decrease the verbosity of Automath books.

—The *block system* allows to share the formal parameters that several global constants have in common (see Section 2.2). Actually, the discussion on instantiation in [dB91] explains that this is more than a mere facility.

—The *paragraph system*, briefly mentioned in [Zan94] and fully explained in [vB77], allows to reuse identifiers avoiding name collisions (see Section 2.3).

—The *abbreviation system* (*i.e.*, the *shorthand facility* [vD94a]) allows to omit some actual parameters in a reference to a global constant (see Section 2.4).

In any case, some aspects of these facilities seem undocumented and thus remain ambiguous to us. As a reasonable way out, we checked how these ambiguities are solved in the QE-GdA knowing that the specification must be correct as it stands.

Contrary to Aut-QE, the formal system $\lambda\delta$ version 3 (see Section 3.1) is an abstract language not supporting any facility in the first place. Therefore, in the first step of our translation, a static analyzer removes the Aut-QE shorthand from the QE-GdA and disambiguates most of its unified binders as we explain in Section 2.5.

The product of this step is a raw version of the $\lambda\delta$-GdA in which the remaining binders are still ambiguous and, thus, need additional processing.

Section 2.1, Section 2.2, Section 2.3, and Section 2.4 present in a single text the most accurate description of Automath concrete syntax and its related facilities.

2.1 Parsing

We recall that a text written in an Automath language (also known as an Automath *book*), is structured as a sequence of lines, each asserting a statement.

The following kinds of statement are available:

```
(contents)
  <book>    ::= [ <line> ]* <EOF>
  <line>    ::= <section> | <context> | <opener> | <decl> | <def>
  <section> ::= "+" [ "*" ]? <id> | "-" <id> | "--"
  <context> ::= <STAR> | <qid> <STAR>
  <opener>  ::= <id> <DEF> <EB> <E> <term>
            |   <id> <E> <term> <DEF> <EB>
            |   "[" <id> <OF> <term> "]"
  <decl>    ::= <id> <DEF> <PN> <E> <term>
            |   <id> <E> <term> <DEF> <PN>
  <def>     ::= <id> <DEF> [ "~" ]? <term> <E> <term>
            |   <id> <E> <term> <DEF> [ "~" ]? <term>
  <term>    ::= <TYPE> | <PROP>
            |   <qid> [ "(" [ <term> [ "," <term> ]* ]? ")" ]?
            |   "[" <id> <OF> <term> "]" <term>
            |   "<" <term> ">" <term>
  <qid>     ::= <id> [ `"` [ <id> ]? [ <PATH> <id> ]* `"` ]?
  <id>      ::= [ "0"-"9" | "A"-"Z" | "a"-"z" | "_" | "'" | "`" ]+
(presentational variants)
  <STAR>    ::= "*" | "@"
  <DEF>     ::= ":=" | "="
  <EB>      ::= "---" | "'eb'" | "EB"
  <PN>      ::= "???" | "'pn'" | "PN" | "'prim'" | "PRIM"
  <E>       ::= "_E" | "'_E'" | ";" | ":"
  <OF>      ::= ":", ","
  <TYPE>    ::= "'type'" | "TYPE"
  <PROP>    ::= "'prop'" | "PROP"
  <PATH>    ::= "-", "."
  <EOF>     ::= ";" | eof
(spaces and comments)
  <space>   ::= [ space | tab | newline ]+
  <comment> ::= [ "#" | "%" ] [ . ]* [ newline | eof ]
            |   "{" [ . ]* "}"
```

Fig. 2. Automath concrete syntax.

`" "`	the enclosed characters	`|`	choice
`` `"` ``	the character "	`[ ]?`	optional
`space tab newline eof`	special characters	`[ ]*`	zero or more
`-`	any character in the specified range	`[ ]+`	one or more
`.`	any character	`[ ]`	bracketing

Fig. 3. Conventions for displaying the concrete syntax.

—Sectioning-related statement. This statement opens or closes a *paragraph* (a better translation would be a *section* as pointed out in [Wie99]). Automath paragraphs are possibly nested named scopes in which the global constants are declared or defined. It should be noted that a previously closed scope can be reopened. Moreover, in a well-formed Automath book, sections are properly nested so each closing statement can only close the last opened section.

—Block opener. This statement introduces a local declaration in a given context. The semantics of context formation is recalled in Section 2.2.

—Global declaration. This is like the block opener but the declaration is global.

—Global definition. This statement defines an identifier as an abbreviation of a term explicitly typed in a given context. There is a way, never used in the QE-GdA, to inhibit the δ-expansion of the identifier for speeding reduction.

An identifier declared or defined by a statement which is not sectioning-related, is termed a *notion*. Moreover, the notions that are not block openers will be termed *global constants*. Automath terms and types are λ-terms of a single syntactic category comprising two sorts, references, unified typed abstractions, and binary applications. References to global constants may have actual parameters and a section indicator acting as a qualifier (see Section 2.3).

In Figure 1 we show the abstract syntax of a typical Automath language. When an identifier `x` is introduced in a book (by abstraction, declaration, or definition), the accompanying term `N` represents its *expected* type (terminology from [Cos96]).

The *degree* of a term stands among the fundamental notions by which the correctness of an Automath book is defined. Generally speaking, it is a number indicating a position in a type hierarchy. A well-established tradition assigns degree $d+1$ to a term T whose type U has degree d. The choice of the terms having degree 1 depends on the type system in discourse. In the present case, we set:

Definition 1. The degree of a term is defined inductively. The degree of `'type'` and `'prop'` is 1; the degree of `x` is $d+1$ where d is the degree of the expected type `N` of `x`; the degree of `[x:N]M` and `<N>M` is the degree of `M`. ▲

As to Aut-QE, correct terms must have degree one, two, or three. Moreover the degree of `N` in the abstraction `[x:N]M` must be two. On the other hand, the degree of `N` in the local declaration (*i.e.*, block opener) `[x:N]C` can be one, or two.

The grammar recognized by our Automath parser is presented in Figure 2 (our notational conventions for displaying grammars are in Figure 3, the monospaced typeface is just presentational). Properly nested comments are accepted. It should be noted that the Automath grammar evolved through time and has many variants [NGdV94], which we try to capture. Our parser recognizes "`_E`" in place of the original underscored "E", anyway this notation does not appear in the QE-GdA.

2.2 Context Chains

The block system is a peculiar feature of concrete Automath languages [NGdV94]. In principle, a global constant has a list of formal parameters that are retrieved by following a chain of block openers, each representing a parameter declaration. To this end, every notion has a *context marker* indicating the start of its chain (*i.e.*, its context). The rules for constructing this chain, follow.

—If the notion has an empty marker, then its chain is empty.

—If the notion has a reference marker pointing to a block opener, then its chain contains the chain of the block opener plus the block opener itself.

—If the notion has no marker and the preceding statement is a block opener, then the intended marker is a reference to it.

—If the notion has no marker and the preceding statement is a global declaration or definition, then the intended marker is the one of that statement (recursively).

The intended meaning of "preceding" in the last two rules becomes unclear when the paragraph system is in effect. Given that the QE-GdA becomes incorrect if

"preceding" is understood literally, we argue that the block system must be aware of the paragraph system somehow. In particular, "preceding" reasonably means "preceding in the same section or in its parent", but may also mean "preceding in the same section fragment or in its parent" (recall that sections can be closed and reopened, thus a section might be divided in many fragments).

The QE-GdA does not help to solve this ambiguity because Jutting always reopens a section with a statement having an explicit context marker.

2.3 Paragraphs

The proper lines of an Automath text using the paragraph system facility, *i.e.*, the lines that are not sectioning-related, are grouped into possibly nested named sections. Furthermore, a *complete index* is assigned to each such line. This is the list of the sections' names containing that line, sorted according to the outermost-to-innermost order. The paragraph system specification requires a *cover* section (named "`1`" in the QE-GdA) enclosing the entire book, so a complete index is never empty. A section can be reopened at the same nesting level but two sections having the same name can not be nested. Once the index of a line is computed, that line receives a URI based on that index [Gui09a]. Note that the *rule for constants* stated in the specification of the paragraph system, implies that different proper lines of a well-formed book always receive different URI's.

A reference r in a line l can have a complete index or an *incomplete index* or no index at all. Such a reference is resolved by computing either the position index of the referred local declaration, or the URI of the referred notion.

The original resolution rules from [vB77] Appendix 2, are given next.

—If r has a complete index j, being the concatenation of the component s before the list j_t, and if the line l has the complete index i, being the concatenation of the list i_h before the component s and before the list i_t, then a constant with r's name is looked up in the section whose index is the concatenation of i_h before j. Such a constant must exist and r receives its URI.

—If r has the incomplete index j and if the line l has the complete index i, then a constant with r's name is looked up in the section whose index is the concatenation of i before j. Such a constant must exist and r receives its URI.

—If r has no index, a declaration with r's name is looked up in the local environment of r. If such a declaration exists, r receives its depth index [dB94b].

—On the other hand, a constant with r's name is looked up in the sections containing the line l, sorted according to the innermost-to-outermost order. Such a notion must exist and r is resolved by receiving its URI.

—If r is a context marker, then r must refer to a block opener otherwise r must refer to a global constant, or to a declaration in r's local environment.

Our implemented processor generalizes the first two rules by extending the search for a notion that should be in a section, say k, to the sections containing k, sorted according to the innermost-to-outermost order. This mechanism agrees with the forth rule and allows to regard a reference without an index as having the complete index of the line in which it occurs. We remark that a reference without an index is resolved first in its local environment and then in the global environment. This

seems to be the originally intended order of precedence because the QE-GdA fails to validate if we reverse this precedence [Wie99].

2.4 Implicit Arguments

The abbreviation system [vD94a] is a facility of some concrete Automath languages including the extension of Aut-QE that Jutting used for the QE-GdA.

This facility works as follows: suppose that a constant c is defined or declared in a context Γ of formal parameters, say $x_1, \ldots, x_n$. Then a reference to c in a subsequent line, say l, generally needs to be applied to n actual parameters and thus appears like $c(t_1, \ldots, t_n)$. Nevertheless, if the context Γ is an initial segment of the context of the notion defined or declared in the line l, where the reference to c appears, this reference is allowed to take less than n actual parameters and the expression $c(t_{m+1}, \ldots, t_n)$ must be interpreted as $c(x_1, \ldots, x_m, t_{m+1}, \ldots, t_n)$. Here we are assuming $m \leq n$, thus all actual parameters may be omitted in some cases.

2.5 Static Disambiguation of Unified Binders

Our static analyzer implements two strategies for disambiguating the binders of the QE-GdA. One strategy is degree-based, while the other is position-based.

Note that both strategies rely on the fact that the QE-GdA is in β-normal form.

In the disambiguation process each binder receives a *layer constant*, which is either "Π", or "λ", or else it receives a *layer variable* in case of ambiguity.

—According to the degree-based strategy [Bro11], we compute the degree of a binder by innermost-to-outermost propagation. Since Aut-QE features three degrees of terms, we argue that a binder of lowest degree (*i.e.*, one) is a "Π", whereas a binder of highest degree (*i.e.*, three) is a "λ". A binder of degree two remains ambiguous. This strategy is not applied to the block-opening binders.

—According to the position-based strategy, a block-opener in the context of a declared constant is a "Π". On the other hand, a block-opener in the context of a defined constant is a "λ", that is β-reduced when that constant is referred to. Moreover, a binder placed along the *spine* of a term representing a type annotation, must be a "Π". The other binders remain ambiguous.

The positioning information is computed by outermost-to-innermost propagation. Therefore, our our static disambiguation procedure is bidirectional.

Finally, our analyzer annotates all binders with their sort (*i.e.*, either "Type", or "Prop") as hints for presenting Π-binders as $\forall$-binders when their sort is "Prop".

3. DYNAMIC ANALYSIS OF THE "GRUNDLAGEN"

The raw version of the $\lambda\delta$-GdA produced by our static analyzer is a global environment of $\lambda\delta$ version 3, a calculus we introduce in Section 3.1 for the first time.

To the end of managing the ambiguous binders, we allow layer variables (say: ϕ, ψ) in abstractors. Therefore, a typical ambiguous abstraction looks like $\lambda_x^{\phi} W.T$.

The raw $\lambda\delta$-GdA is analyzed by applying a validation procedure that produces a system of constraints on the layer variables (see Section 3.5). Once this system is solved (also by correcting some points of the QE-GdA as we explain in Section 3.6), the proper $\lambda\delta$-GdA, without layer variables, is mapped to the CC-GdA, *i.e.*, our final outcome. This is a single user-level script that can be processed by Coq 8.4.3.

index:	k,n	::=	integer	$0 \leq k, n < \infty$
name:	u,x	::=	identifier	Barendegt's convention is assumed
layer:	e	::=	ext. integer	$0 \leq e \leq \infty$
term:	T,U,V,W	::=	$\star k$	sort of index k
		|	$\$u$	reference to global name u
		|	$\#x$	reference to local name x
		|	$\delta_x V.T$	local abbreviation $(x \coloneqq V)$ in T
		|	$\lambda^e_x W.T$	local abstraction in layer e of $(x : W)$ in T
		|	$@V.T$	applicative term $(T\ V)$
		|	$ⓒU.T$	type-annotated term $(T : U)$
local env.:	K,L	::=	$\star$	empty environment
		|	$L.\delta_x V$	local definition $(x \coloneqq V)$ after L
		|	$L.\lambda^e_x W$	local declaration $(x : W)$ in layer e after L
global env.:	F,G	::=	$\star$	empty environment
		|	$G.\delta_u V$	global definition $(u \coloneqq V)$ after G
		|	$G.\lambda_u W$	global declaration $(u : W)$ after G

Fig. 4. The abstract syntax of terms and environments.

$$\frac{}{G_1.\lambda_u W.G_2, L \vdash \$u \overset{0}{\twoheadrightarrow}_h \$u}\,\rho\$ \qquad \frac{}{G, L_1.\lambda^e_x W.L_2 \vdash \#x \overset{0}{\twoheadrightarrow}_h \#x}\,\rho\#$$

$$\frac{}{G, L \vdash \star k \overset{n}{\twoheadrightarrow}_h \star h^n(k)}\,s \qquad \frac{G, L \vdash V_1 \overset{0}{\twoheadrightarrow}_h V_2 \quad G, L \vdash T_1 \overset{n}{\twoheadrightarrow}_h T_2}{G, L \vdash @V_1.T_1 \overset{n}{\twoheadrightarrow}_h @V_2.T_2}\,\nu$$

$$\frac{G, L.\delta_x V \vdash T_1 \overset{n}{\twoheadrightarrow}_h T_2}{G, L \vdash \delta_x V.T_1 \overset{n}{\twoheadrightarrow}_h \delta_x V.T_2}\,\sigma \qquad \frac{G, L \vdash W_1 \overset{0}{\twoheadrightarrow}_h W_2 \quad G, L.\lambda^{e-n}_x W_1 \vdash T_1 \overset{n}{\twoheadrightarrow}_h T_2}{G, L \vdash \lambda^e_x W_1.T_1 \overset{n}{\twoheadrightarrow}_h \lambda^{e-n}_x W_2.T_2}\,x$$

$$\frac{G_1, L \vdash V_1 \overset{n}{\twoheadrightarrow}_h V_2}{G_1.\delta_u V_1.G_2, L \vdash \$u \overset{n}{\twoheadrightarrow}_h V_2}\,\delta\$ \qquad \frac{G, L_1 \vdash V_1 \overset{n}{\twoheadrightarrow}_h V_2}{G, L_1.\delta_x V_1.L_2 \vdash \#x \overset{n}{\twoheadrightarrow}_h V_2}\,\delta\#$$

$$\frac{G, L \vdash \delta_x V_2.@V_1.T_1 \overset{n}{\twoheadrightarrow}_h T_2}{G, L \vdash @V_1.\delta_x V_2.T_1 \overset{n}{\twoheadrightarrow}_h T_2}\,\theta \qquad \frac{G, L \vdash \delta_x(ⓒW.V).T_1 \overset{n}{\twoheadrightarrow}_h T_2}{G, L \vdash @V.\lambda^e_x W.T_1 \overset{n}{\twoheadrightarrow}_h T_2}\,\beta$$

$$\frac{G, L \vdash T_1 \overset{n}{\twoheadrightarrow}_h T_2}{G, L \vdash \delta_x V.T_1 \overset{n}{\twoheadrightarrow}_h T_2}\,\zeta \qquad \frac{G, L \vdash T_1 \overset{n}{\twoheadrightarrow}_h T_2}{G, L \vdash \lambda^e_x W.T_1 \overset{n}{\twoheadrightarrow}_h T_2}\,\upsilon$$

$$\frac{G, L \vdash T_1 \overset{n}{\twoheadrightarrow}_h T_2}{G, L \vdash ⓒU.T_1 \overset{n}{\twoheadrightarrow}_h T_2}\,\epsilon \qquad \frac{G, L \vdash U_1 \overset{n}{\twoheadrightarrow}_h U_2}{G, L \vdash ⓒU_1.T \overset{n+1}{\twoheadrightarrow}_h U_2}\,e$$

$$\frac{G_1, L \vdash W_1 \overset{n}{\twoheadrightarrow}_h W_2}{G_1.\lambda_u W_1.G_2, L \vdash \$u \overset{n+1}{\twoheadrightarrow}_h W_2}\,l\$ \qquad \frac{G, L_1 \vdash W_1 \overset{n}{\twoheadrightarrow}_h W_2}{G, L_1.\lambda^e_x W_1.L_2 \vdash \#x \overset{n+1}{\twoheadrightarrow}_h W_2}\,l\#$$

Side conditions: $n < e$ for β and $e \leq n$ for υ; x not free in T_1 for ζ and υ.

Fig. 5. The proposed rt-transition system.

The apparatus for validating in $\lambda\delta$ version 3 derives essentially from [Gui10], which refers to a previous version of the calculus, and consists of a reduction machine (Section 3.2), two controllers (Section 3.3) asserting applicability and convertibility in context, and a validator (Section 3.4) implementing top-level verification.

3.1 The Formal System $\lambda\delta$ Version 3: "To Π ... and Beyond"

The $\lambda\delta$ family is a sequence of typed λ-calculi taking their features from several formal systems. We stress that this family is not related intentionally to any other

$$\frac{}{G, L \vdash \star k \,!\,_h}\,S \qquad \frac{G, L \vdash V \,!\,_h \quad G, L.\delta_x V \vdash T \,!\,_h}{G, L \vdash \delta_x V.T \,!\,_h}\,F \qquad \frac{G, L \vdash W \,!\,_h \quad G, L.\lambda^e_x W \vdash T \,!\,_h}{G, L \vdash \lambda^e_x W.T \,!\,_h}\,C$$

$$\frac{G_1, L \vdash V \,!\,_h}{G_1.\delta_u V.G_2, L \vdash \$u \,!\,_h}\,D\$ \qquad \frac{G, L_1 \vdash V \,!\,_h}{G, L_1.\delta_x V.L_2 \vdash \#x \,!\,_h}\,D\#$$

$$\frac{G_1, L \vdash W \,!\,_h}{G_1.\lambda_u W.G_2, L \vdash \$u \,!\,_h}\,I\$ \qquad \frac{n, G \vdash L_1 \,!\,_h W}{G, L_1.\lambda^e_x W.L_2 \vdash \#x \,!\,_h}\,I\#$$

$$\frac{G, L \vdash U \,!\,_h \quad G, L \vdash T \,!\,_h \quad G, L \vdash U \overset{0}{\twoheadrightarrow}_h U_0 \quad G, L \vdash T \overset{1}{\twoheadrightarrow}_h U_0}{g, L \vdash \copyright U.T \,!\,_h}\,T$$

$$\frac{G, L \vdash V \,!\,_h \quad G, L \vdash T \,!\,_h \quad G, L \vdash V \overset{1}{\twoheadrightarrow}_h W_0 \quad G, L \vdash T \overset{n}{\twoheadrightarrow}_h \lambda^{e+1} W_0.U_0}{G, L \vdash @V.T \,!\,_h}\,\mathrm{A}*$$

$$\frac{}{\vdash \star \,!\,_h}\,S\$ \qquad \frac{\vdash G \,!\,_h \quad G, \star \vdash V \,!\,_h}{\vdash G.\delta_u V \,!\,_h}\,F\$ \qquad \frac{\vdash G \,!\,_h \quad G, \star \vdash W \,!\,_h}{\vdash G.\lambda_u W \,!\,_h}\,C\$$$

Side condition for A$*$: Figure 6(υ) not allowed when reducing the spine of T (forth premise).

Fig. 6. The proposed validity of terms and global environments.

$$\frac{G_1, L \vdash V :_h W}{G_1.\delta_u V.G_2, L \vdash \$u :_h W}\,D\$ \qquad \frac{G, L_1 \vdash V :_h W}{G, L_1.\delta_x V.L_2 \vdash \#x :_h W}\,D\#$$

$$\frac{G_1, L \vdash W :_h V}{G_1.\lambda_u W.G_2, L \vdash \$u :_h W}\,I\$ \qquad \frac{G, L_1 \vdash W :_h V}{G, L_1.\lambda^e_x W.L_2 \vdash \#x :_h W}\,I\#$$

$$\frac{}{G, L \vdash \star k :_h \star(h(k))}\,S \qquad \frac{G, L \vdash T :_h U \quad G, L \vdash U :_h W}{G, L \vdash \copyright U.T :_h \copyright W.U}\,T$$

$$\frac{G, L \vdash V :_h W \quad G, L.\delta_x V \vdash T :_h U}{G, L \vdash \delta_x V.T :_h \delta_x V.U}\,F \qquad \frac{G, L \vdash W :_h V \quad G, \lambda^e_L x.W \vdash T :_h U}{G, L \vdash \lambda^e_x W.T :_h \lambda^{e-1}_x W.U}\,C$$

$$\frac{G, L \vdash V :_h W \quad G, L \vdash \lambda^1_x W.T :_h U}{G, L \vdash @V.\lambda^1_x W.T :_h U}\,A0 \qquad \frac{G, L \vdash V :_h W \quad G, L \vdash T :_h \lambda^{e+1}_x W.U}{G, L \vdash @V.T :_h @V.\lambda^{e+1}_x W.U}\,A1$$

$$\frac{G, L \vdash T :_h U \quad G, L \vdash @V.U :_h W}{G, L \vdash @V.T :_h @V.U}\,P*$$

$$\frac{G, L \vdash T :_h U_1 \quad G, L \vdash U_2 :_h W \quad G, L \vdash U_1 \overset{0}{\twoheadrightarrow}_h U \quad G, L \vdash U_2 \overset{0}{\twoheadrightarrow}_h U}{G, L \vdash T :_h U_2}\,E$$

Fig. 7. The conjectured type assignment.

$$\frac{G, L \vdash V \,!\,_h \quad G, L \vdash T \,!\,_h \quad G, L \vdash V \overset{1}{\twoheadrightarrow}_h W_0 \quad G, L \vdash T \overset{1}{\twoheadrightarrow}_h \lambda^{e+1} W_0.U_0}{G, L \vdash @V.T \,!\,_h}\,\mathrm{A1}$$

Side condition for A1: Figure 6(υ) not allowed when reducing the spine of T (forth premise).

Fig. 8. The proposed restricted applicability condition.

system having (variations of) the symbols λ and δ in its name or syntax. Examples include (but are not limited to): λ-δ [Chu41], $\Delta\Lambda$ [dB94a, dB93], λ_Δ [RS94], $\lambda\Delta$ [RP04], λD [NG14], $C\lambda\xi$ [dB78], and [Ned79, Ned80, BKN96].

The first calculus of the $\lambda\delta$ family, introduced in [Gui09b], extends the system λ^λ [dG93] taking some features from [dG91]. Another calculus, introduced in [Gui15], adds the applicability condition of the system Λ_∞ [vB94b]. The calculus we are presenting here as $\lambda\delta$ version 3, is designed for the dynamic analysis of the QE-GdA and adds *type inclusion by reduction* to the previous calculi of the family.

Using this device for the formation of universes, that generalizes *sort inclusion* of the system $\lambda\lambda$ [dV94], our calculus can integrate Λ_∞ with Aut-QE and CC.

The calculus comprises three components: the generative grammar (Definition 2), the transition system (Definition 3), and the validity rules (Definition 4).

We note that the primitive notion for defining the syntactically correct terms is the validity judgment rather than the type judgment. We justify this approach in [Gui15] where we remark that the formal treatment of validity can be more convenient than that of typing. In fact, a type judgment conveys the information that its subject (say T) is valid and, in addition, shows a specific type U of T. Nevertheless, the choice of U is sometimes pointless and always arbitrary given that types are not unique but are assigned up to conversion in the present setting.

Its important to stress that the system we are proposing still lacks a theoretical study. In particular, the rules of the type judgment have not been established yet. However, Figure 7 shows our conjecture for these rules. The reader should note that the rules of λ^λ are included, as well as the *pure* type rule for application advocated in [dB91]. In order to reach such a type system, we might need to correct the present definitions slightly. A disadvantage of the conjectured type system is that the single rule Figure 6(A$*$) splits in the three rules Figure 6(A0, A1, P$*$).

Definition 2. We define terms and environments in Figure 4. References occur by name. We assume Barendegt's convention in that the same name does not occur free and bound in the same term or judgment, nor is bound more than once.
Layers are integers extended with ∞. For $n < \infty$ we set: $\infty - n = \infty + n = \infty$. ▲

Definition 3. Given a function h chosen at will as long as $\forall k.\ k < h(k)$, and given an integer n such that $0 \leq n < \infty$, the predicate $G, L \vdash T_1 \overset{n}{\twoheadrightarrow}_h T_2$ defined in Figure 5 states that T_1 computes to T_2 in our rt-transition system [vD94b].
Transition occurs in the context of G and L, and with respect to h and n.
The system comprises structural schemes, reduction schemes (the r-transitions), and type inference schemes (the t-transitions). Thus, the *height* n indicates the desired difference of degree between T_1 and T_2 occurring because of the t-transitions. Firstly, the structural steps are: $\rho\$$, $\rho\#$, s for $n = 0$ (termed $\rho\star$), ν, σ, and x for $n = 0$ (termed ξ). Secondly, the reduction steps are: β (function application), $\delta\$$ (global expansion), $\delta\#$ (local expansion), ζ (abbreviation removal), θ (commutation of application-abbreviation pair as in [CH00]), ϵ (annotation removal), and our υ (λ^0-abstraction removal) that generalizes *sort inclusion*. Thirdly, the type inference steps are: s for $n > 0$ (sort typing), $l\$$ (global reference typing), $l\#$ (local reference typing), x for $n > 0$ (abstraction typing) and e (annotation typing). ▲

Definition 4. The predicate $G, L \vdash T\,!_h$ defined in Figure 6 states the validity of T (*i.e.*, its syntactical correctness) with respect to h in the context of G and L.
In particular, Figure 6(A$*$) states the *extended* applicability condition.
The predicate $\vdash G\,!_h$ defined in Figure 6 states the validity of G w.r.t. h.
A type judgment is defined by setting: $G, L \vdash T :_h U$ iff $G, L \vdash \copyright U.T\,!_h$. ▲

It is a fact that sort inclusion and β-contraction yield a non-confluent critical pair when applied to the same λ-abstraction. In this respect, Automath's approach [Zan94] is to apply sort inclusion only when β-contraction is not applicable.

On the other hand, the key idea behind $\lambda\delta$ version 3 is to apply sort inclusion and β-contraction to different λ-abstractions. To this end, a λ^e-abstraction is provided for each value of the integer *layer* e in the range $0 \le e \le \infty$. In this setting, sort inclusion applies to λ^0-abstractions in the form $G, L \vdash \lambda^0_x W.T \overset{0}{\twoheadrightarrow}_h T$ (x not free in T) implied from Figure 5(υ), while β-contraction applies to λ^{e+1}-abstractions as we imply from Figure 5(β) when $n = 0$. Thus, a λ^{e+1}-abstraction is a head normal form and as such it appears in Figure 6(A$*$), that states our applicability condition.

The infinitely many λ^e-abstractions are linked by stating that a λ^e-abstraction is canonically typed by a λ^{e-1}-abstraction, as we imply from Figure 5(x) when $n = 1$.

In this respect, λ^∞ is typed by λ^∞ (as one expects), and λ^0 is typed by λ^0.

Nevertheless, the strict extension condition $k < h(k)$ of Definition 3, applied to Figure 5(s) when $n = 1$, yields that the sort of index k is typed by a sort of higher index. Therefore, a term is never typed by itself (as the reader might fear).

As of λ^1, the assumption $G, L \vdash T \overset{1}{\twoheadrightarrow}_h \star k$ (*i.e.*, T is typed by the sort of index k) yields by Figure 5(υ) the conclusion $G, L \vdash \lambda^1_x W.T \overset{1}{\twoheadrightarrow}_h \star k$ (*i.e.*, $\lambda^1_x W.T$ is typed by the same sort). The situation is explained by observing that $\lambda^1_x W.T$ types with $\lambda^0_x W.\star k$ which υ-reduces to $\star k$, and suggests that λ^1 corresponds to Π in CC.

This digression yields with no surprise that λ^2 must stand for λ in CC.

Stepping up the λ^e-hierarchy, we meet λ^3, which is typed by λ^2 as in the systems of the Aut-4 family [dB94c], as well as the abstractions advocated in [KBN99].

As to λ^0, since it types Π, we agree to term it *hyper* Π, which literally means *beyond* Π. Its role in Aut-QE is that of constructing some *quasi-expressions*.

In principle, υ-contraction can include big universes into small ones, so, eventually, restrictions might apply to the term W of Figure 5(υ) in order to prove strong normalization and to avoid inconsistency. These restrictions will influence the λ^e-abstractions with $0 \le e < \infty$, while the λ^∞-abstractions will not be influenced. In fact, we see that layer 0 cannot be reached by iterated typing from layer ∞.

Thus, $\lambda\delta$ version 3 accounts for the distinction between *instantiation* and *abstraction* [dB91] by which block openers correspond to λ^∞-abstractions, whereas functional abstractions correspond to λ^e-abstractions with $0 < e < \infty$.

However, in order to validate the QE-GdA in CC, block openers must be represented in layer 1 and 2. We will discuss the consequences of this issue in Section 5.

The restriction of $\lambda\delta$ version 3 to layer ∞ is essentially the calculus we presented in [Gui15]. Remarkably, the calculus we present in this article is a minimal-impact extension of that system, which adds just one new rule: υ-contraction.

As of our applicability condition, the extended form stated by Figure 6(A$*$), where n can have any value, is the one of Λ_∞: a calculus featuring more than three degrees of valid terms. However, in the subsystem for representing Aut-QE and CC, where only three degrees of valid terms are available, taking 1 for n suffices. Thus, we obtain the *restricted* applicability condition shown in Figure 8.

In this respect, the calculus we presented in [Gui09b] is essentially $\lambda\delta$ version 3 restricted to layer ∞ and equipped just with the restricted applicability condition.

We observe that the author of the QE-GdA relies on extended applicability to verify the constant `ande2"l-r"`, where a variable for a predicate is unified with its universal quantification four times. This issue is discussed in Section 3.6.

Looking at the premise $G, L \vdash T \overset{n}{\twoheadrightarrow}_h \lambda^{e+1} W_0.U_0$ of Figure 6(A$*$), we see that W_0

	term	$n=0$ / $d=1$	$n>0$ / $d>1$	n / d	restricted mode / extended mode
01r	$\star h$	stop	$\rightarrow$s	dec.	
01x		stop			
02r	$\textcircled{c}U.T$	$\rightarrow\epsilon$	$\rightarrow$e	dec.	
02x		$\rightarrow\epsilon$		as is	
03r	$\lambda^e W.T$ before up.	as is	$\rightarrow$x	as is	to *after up.*
03x		as is		$d \wedge e$	
04	$\lambda^e W.T$ after up. with empty stack	stop		as is	restarts with $\rightarrow\xi$
05	$\lambda^{e+1} W.T$ after up. with $@V$ on stack	$\rightarrow\beta$		as is	also if $e+1$ is ϕ
06	$\lambda^0 W.T$ after up. with $@V$ on stack	error			invalid application
07	$\delta_x V.T$	$\rightarrow\sigma$		as is	push $\delta_x V$ on env.
08	$@V.T$	$\rightarrow\nu$		as is	push $@V$ on stack
09	dangling $\$x$ or $\#x$	error			invalid reference
10	$\$u$ with $\delta_u V$ on env.	stop		as is	restarts with $\rightarrow\delta\$$
11	$\#x$ with $\delta_x V$ on env.	$\rightarrow\delta\#$		as is	
12	$\$u$ with $\lambda_u W$ on env.	stop	$\rightarrow$l$\$$	dec.	
13	$\#x$ with $\lambda_x^{e+1} W$ on env.	stop	$\rightarrow$l$\#$	dec.	also if $e+1$ is ϕ
14	$\#x$ with $\lambda_x^0 W$ on env.	error	$\rightarrow$l$\#$	dec.	failure of υ

(1) $\rightarrow$s is: $\star k \rightarrow \star h(k)$
(2) $\rightarrow$x is: $\lambda^e W.T \rightarrow \lambda^{e-n} W.T$ for $0 < n$
(3) $\rightarrow\xi$ is: push $\lambda_x^e W$ on env.
(4) column n and d: value after step (dec.: decrement if positive, $\wedge$: minimum)
(5) single line: same operation in both modes

Fig. 9. Operational semantics of the RTM.

(*i.e.*, the expected type of the application argument) may depend on n. However, several side conditions can be set if we desire to avoid this dependence.

Our choice to exclude υ-contractions when deriving the premise, is implied by the use of sort inclusion in [Zan94] and has some advantages. On the one hand, υ-contraction remains fully general. On the other hand, there is no need to check the side condition of υ-contractions during the mechanical derivation of the premise.

3.2 An Overview of the Reduction and Type Machine

Mechanical validation in the $\lambda\delta$ family is based on the Reduction and Type Machine (RTM). This is an abstract machine of the K family [Kri07], defined first in [Gui10], that computes the weak head normal forms by rt-reduction according to Figure 5.

The RTM is a *controlled* machine, in that it can stop before global δ-expansions and υ-contractions, and then it can be restarted by the calling controller. In fact, these reductions are better managed on the controller's side (see Section 3.3).

Our verification procedure involves the RTM to test the next conditions:

(1) convertibility for type annotation: premises 3 and 4 of Figure 6(T);
(2) convertibility for application: premise 3 of Figure 6(A$*$) and of Figure 8(A1);
(3) restricted applicability: premise 4 of Figure 8(A1).
(4) extended applicability: premise 4 of Figure 6(A$*$);

In the last case the computation's height n is not known in advance. In the other cases, instead, this height is set to 0 or 1. Thus, the RTM runs in two modes.

—When the the caller specifies a value for n, the RTM runs in the restricted mode, suitable for cases (1) to (3). In this mode the e-step (when applicable) is preferred to ϵ-step. This means that, in case of type inference, we use expected types if we know them. The reader should note the type annotation in Figure 5(β).
Operating on a valid term, the RTM detects an error just upon failure of the side condition for the υ-step, which the machine checks with a lazy policy.

—When the caller does not specify a value for n, the RTM needs an upper bound d for it (∞ is allowed) and runs in the extended mode, suitable for case (4).
In this mode, targeted at extended applicability, the RTM looks for a λ^{e+1}-abstraction by repeatedly increasing the height n of the computation, or else it stops as soon as such an abstraction cannot be found in the range $0 \leq n < d$.
We remark that, on valid terms, this problem is decidable even for $d = \infty$.
The RTM increases n just when forced by an l-step, so s-steps, e-steps and x-steps are disabled. Anyway, care is taken to ensure the side condition for $\rightarrow\beta$.
Suppose the RTM finds the β-redex $@V.\lambda^e_x W.T_0$ and computes it. If the height of this computation is increased by n because of following l-steps, the redex eventually becomes $@V.\lambda^{e-n}_x W.T_n$ and remains valid only if $n < e$. The RTM ensures this condition by updating its height's upper bound to at most e on encountering a λ^e-abstraction. Thus, we explain the need for the upper bound.

The raw QE-GdA coming from the static analyzer, is verified under the restricted applicability condition and contains abstractions with layer variables. Thus, the RTM is instructed to deal with these variables in the restricted mode.

In particular, when a decision must be made on $\lambda^\phi_x W$ whether to consider $\phi > 0$ or $\phi = 0$ for an unknown ϕ, the first alternative is chosen being the safest. In fact, choosing $\phi = 0$ may lead the RTM to an error condition (either invalid application, or failure of υ-step). It follows that on $\lambda^\phi_x W.T$, the β-step is taken whenever possible, in accordance with the original verification algorithm for Aut-QE [Zan94].

Figure 9 displays, informally but precisely, the operational semantics of the RTM updated for $\lambda\delta$ version 3. The extended mode is included for completeness.

We stress that being a machine of the K family, the RTM avoids ζ-contractions, which are known as *delifting steps* following a well-established terminology.

3.3 An Overview of the Controllers

The RTM's are operated by two controllers: the *comparator* and the *applicator*.

The comparator asserts the convertibility between an expected type U and the inferred type of a term V. To this end, it starts a machine on U with $n = 0$, and a machine on V with $n = 1$. So, both machines run in the restricted mode.

The conversion test occurs by levels, *i.e.*, by repeated comparison of weak head normal forms. The comparison policy follows [Gui10] and is given next.

(1) Two sorts are compared by their index. The arguments stacked by the two machines are not considered since the RTM's run on valid terms.

(2) Two references to local abstractions are compared by their level, *i.e.*, not by their depth, thus these references are not relocated. Information on the level of each reference is provided by the respective RTM. In case of match, we assert the convertibility of the arguments stacked by the two machines.

(3) Two references to global declarations are compared by URI. In case of match, we assert the convertibility of the arguments stacked by the two machines.
(4) Two references to global definitions are compared by URI. In case of match, we test the convertibility of the arguments stacked by the two machines and, if this test fails, we δ-expand both definitions. In case of mismatch, we δ-expand the *older* definition according to an *age* system we shall explain.
(5) A reference to a global definition compared to any other term, is δ-expanded.
(6) An abstraction in layer 0 is υ-contracted by restarting its machine.
(7) Two abstractions are compared by their layer and, in case of match, we assert the convertibility of their arguments. Variable layers are accepted.
(8) A variable layer abstraction compared to any other term is υ-contracted.
The earlier definition of this controller [Gui10] was asymmetric in this clause. Following [Zan94] too closely, we contracted just an abstraction coming from the term V compared with a sort coming from U. Moreover, we disabled this clause when leaving the *spine* of V to avoid an evident inconsistency [Gui09a], which disappears by restricting υ-contraction to abstractions in layer 0.

It is a known fact that the QE-GdA is validated faster if we limit δ-expansions during convertibility tests. To this end, the comparator implements *age-controlled* δ-expansions [Zan94]. In particular, a progressively increasing integer $\$u$ is assigned to each constant u after its static analysis. Thus, constants become totally ordered.

The lines of the QE-GdA appear in order of dependence, so a constant u_1 processed before a constant u_2, *i.e.*, satisfying $\$u_1 < \$u_2$, cannot depend on u_2. So, when we compare a reference to u_1 with a reference to u_2, it is safe to δ-expand just the reference to u_2. In this respect, u_2 is the *older* constant of the two.

The second controller, the applicator, tests the applicability of a term T by starting a machine on T with $n = 1$ or with $d = \infty$ depending on the mode desired by the user. The test succeeds if the machine stops on $\lambda_x^{e+1} W_0.U_0$ where the layer may be a variable. In particular, the controller operates thus:

(1) On an abstraction, its layer is tested for positivity.
(2) On a reference to a definition, the machine is restarted, so a δ-expansion occurs.
(3) in the other cases the test fails.

The machine is not restarted on λ^0-abstractions, thus υ-contractions are avoided as the side condition for applicability specifies. See Figure 6(A$*$) and Figure 8(A1).

3.4 An Overview of the Validator

Indeed, the validator is the simpler component of our verification system. To some extent, it follows [Gui10] but, remarkably, we replace the canonical type synthesizer with a procedure to assert the validity relation $\vdash G\,!_h$ of Figure 6.

As a result, the overall validation of the $\lambda\delta$-GdA becomes 1% faster on average (the type synthesizer is still available, thus we can compare the two approaches).

Our point is that computing the canonical type of a term, is more expensive than just asserting its existence. Contrary to the rules of canonical typing, the rules of validity are relocation-free, *i.e.*, they do not involve *lifting*, following a well-established terminology. Thus, in the end, we achieve the long awaited fully relocation-free verification process we advocated in [Gui09a].

Looking at Figure 6(A∗) and Figure 8(A1), the validator asserts the applicability of T to V by calling the applicator on T, which provides a RTM, say M, ready on $\lambda^{e+1}W_0.U_0$. Secondly, it calls the comparator on W_0 and on V, using M in its current state for W_0. On the contrary, the RTM for V has an initial state.

3.5 Dynamic Disambiguation of Unified Binders

When layer variables are allowed, the discussed validation procedure produces a set of constraints on such variables. These constraints are of four kinds.

(1) The constraint $\phi - n = \psi$ is generated by the RTM on an x-step from the abstraction $\lambda^{\phi}W.T$ to the abstraction $\lambda^{\psi}W.T$. See line 03r of Figure 9.
(2) The constraint $\phi = \psi$ is generated by the comparator matching two abstractions $\lambda^{\phi}W_1.T_1$ and $\lambda^{\psi}W_2.T_2$ See Clause (7) of the comparator in Section 3.3.
(3) The constraint $\phi = 0$ is generated by the comparator υ-reducing the abstraction $\lambda^{\phi}W.T$. See Clause (8) of the comparator in Section 3.3.
(4) The constraint $\phi > 0$ is generated by the RTM when it β-reduces $\lambda^{\phi}W.T$. Moreover, it is generated when asserting the applicability of a term with functional structure $\lambda^{\phi}W_0.U_0$ See Clause (1) of the applicator in Section 3.3.

Whenever a constraint is issued, the system of known constraints is reduced by repeated substitution. Thus, inconsistencies are discovered as soon as possible.

Notably, a consequence of static disambiguation (see Section 2.5), is that the unified binders of the $\lambda\delta$-GdA can be disambiguated constant by constant. In fact, all layer variables remaining in a constant u after static analysis are determined after u is validated, *i.e.*, without checking the subsequent references to u.

This means that layer variables can be reused after each constant is validated, and that the system of constraints can be kept small during validation.

We would like to stress that both static disambiguation strategies must be used in order to achieve this result, *i.e.*, they disambiguate distinct sets of binders.

3.6 A Posteriori Static Corrections

As is stands originally, the QE-GdA fails to validate both in the $\lambda\delta$ family and in CC, since two constants require formal η-reduction on Π (*i.e.*, an inferred type $\lambda^1_x W.(@x.T)$ must reduce to an expected type T). They are `t2"l-some"` and `th2"l-r-imp"`. Nevertheless, these reductions can be avoided by following a suggestion due to van Daalen and reported in [vB77], by which we apply a ∀-introduction to a predicate symbol in two constants: `all"l"` and `imp"l-r"` (see Figure 10).

With these corrections, our dynamic analysis of the QE-GdA reports 20 inconsistencies in its layer constraint system. Four of these are located in the constant `ande2"l-r"` and state that sort inclusion is required on Π (*i.e.*, υ-contraction is required in λ^1) four times. We highlight the problem in [Gui09a], noting that `ande2"l-r"` requires the *pure* type inference rule for function application [dB91], corresponding to the extended applicability condition. See Figure 6(A∗).

In [Gui15] we note that, sometimes, extended (*i.e.*, Λ_∞-like) applicability reduces to restricted (*i.e.*, CC-like) applicability by applying λ^{e+1}-introductions.

In the case of the QE-GdA, we invoke `imp"l-r"` to apply a ∀-introduction to a predicate `b` passed as a type and occurring four times (see Figure 10).

```
 all"l"
−all:=p:'prop'
+all:=[x:sigma]<x>p:'prop'
```

```
 imp"l-r"
−imp:=b:'prop'
+imp:=[x:a]<x>b:'prop'
```

```
 ande2"l-r"
−b@[a1:and(a,b)]
−ande2:=<ande1(a,b,a1)>ande2"l"(a,b,a1):<ande1(a,b,a1)>b
+b@[a1:and(a,imp(a,b))]
+ande2:=<ande1(a,imp(a,b),a1)>ande2"l"(a,imp(a,b),a1):<ande1(a,imp(a,b),a1)>b
```

```
 some"l"
−some:=not(non(p)):'prop'
+none:=all(sigma,non(p)):'prop'
+some:=not(none(p)):'prop'
```

∓`non` replaced by `none` in 5 "block openers" and in the constants:
`th1"l-some"`, `th3"l-some"` (two times), `th5"l-some"`, `empty"l-e-st"`, `t5"l-e-st-isset"` (two times), `t13"l-e-st-eq-landau-n-327"`, and `t38"l-e-st-eq-landau-n-327"`

Marks: − (old text), + (new text), ∓ (multiple replacement)

Fig. 10. Static corrections to the QE-GdA.

name:	x	::= identifier	Barendegt's convention is assumed
term:	M,N	::= $\mathrm{Type}, \mathrm{Prop}$	sorts
		| x	reference to name x
		| $\Pi_x N.M$	dependent function space from N to M
		| $\lambda_x N.M$	abstraction of $(x:N)$ in M
		| $(M\ N)$	applicative term
		| $(M:N)$	type-annotated term
environment:	Γ	::= ϵ	empty environment
		| $\Gamma.(x:=M)$	definition of x after Γ
		| $\Gamma.(x:N)$	declaration of x after Γ

Fig. 11. The abstract syntax of the refined Calculus of Constructions.

After the correction, just 12 inconsistencies remain. The first one is located in the constant `some"l"`, where the predicate `non(p)` is passed as a proposition.

We cannot apply a ∀-introduction to `non(p)` in its definition since some constants use it a predicate indeed (they are: `somei"l"`, `t1"l-some"`, `t2"l-some"`, `th1"l-some"`, `t3"l-some"`, `t4"l-some"`, and `th2"l-some"`). So, we introduce a new constant `none` for the ∀-quantified `non(p)` and we replace `non` with `none` where required (see Figure 10). Note that `none` is indeed the counterpart of `some`.

This correction solves all inconsistencies. In the end, we add 21 ∀-introductions to the original QE-GdA to obtain the CC-GdA. This result agrees with [Bro11] stating that the QE-GdA validates in CC just by formal η-expansion.

In this respect, we stress that Brown solves these inconsistencies with a poorly documented automated procedure that adds more ∀-introductions to the original text than we do by manual operation (*i.e.*, 25 ∀-introductions rather than 21).

	Aut-QE	$\lambda\delta$ version 3		Refined CC
Terms (M, N)	`'type' 'prop'`	$\star 0$ $\star 1$		Type Prop
	`x`	$\#x$ or $\$x$		x
	`<N>M`	$@N.M$		$(M\ N)$
	`[x:N]M`	$\lambda_x^\phi N.M$	$\lambda_x^1 N.M$	$\Pi_x N.M$
			$\lambda_x^2 N.M$	$\lambda_x N.M$
Block openers (in list C)	`[x:N]`	$\lambda_x^\phi N$	$\lambda_x^1 N$	$\Pi_x N$
			$\lambda_x^2 N$	$\lambda_x N$
Alternative translation		$\lambda_x^\infty N$		Not supported
Statements	`C@x:='prim':N`	$\lambda_x(C.N)$		$(x : C.N)$
	`C@x:=M:N`	$\delta_x(C.©N.M)$		$(x := C.(M : N))$

Note: the contents of metavariables M, N, C are translated recursively.

Fig. 12. The translation scheme for the QE-GdA.

QE-GdA	CC-GdA
`+l`	
`@[a:'prop'][b:'prop']`	
`imp:=[x:a]b:'prop'`	$(l\_imp := \lambda_a \mathrm{Prop}.\lambda_b \mathrm{Prop}.(\Pi_x a.b : \mathrm{Prop}))$

Fig. 13. The first three lines of the QE-GdA translated in the refined CC.

4. TRANSLATION OF THE "GRUNDLAGEN"

In this section we want to discuss how the QE-GdA is mapped in $\lambda\delta$ version 3 and then in CC. The present translation aims at respecting the distinction between declared constants and defined constants. Moreover, and contrary to [Gui09a], it aims at preserving the shared structure of block openers in defined constants (*i.e.*, block openers are not replicated in the expected type of constants).

For this purpose, it is convenient to equip the original CC with context entries for definitions, of the form $(x := N)$, and with type-annotated terms, of the form $(M : N)$, as we show in Figure 11. We stress that these extensions do not alter the expressive power of CC, and are accepted by Coq 8.4.3 without modifications.

In particular, the reader can derive the rules for the refined CC from [Coq15].

The translation of an Aut-QE text into the refined CC through $\lambda\delta$ version 3 is straightforward as we see in Figure 12, given that the three systems have many common constructions. An example from the QE-GdA is in Figure 13.

The proper translation of Aut-QE block openers would involve λ^∞-abstractions, but the refined CC does not support them. This means that the constant `imp"l"` would be better translated into $\lambda\delta$ version 3 as: $\delta_{l\_imp}(\lambda_a^\infty \star 1.\lambda_b^\infty \star 1.©\star 1.\lambda_x^1 a.b)$.

It is important to stress that in translating $\lambda\delta$ version 3 into the refined CC, we assume that only λ^1-abstractions and λ^2-abstractions occur. That is to say that we assume that the text to be translated actually comes from Aut-QE.

The question remains open if and how a Pure Type System can approximate $\lambda\delta$ version 3. We would approach this problem by looking for a suitable generalization of the Systems approximating Aut-4 and Aut-QE (see for instance [KN94, Bar93]).

We validate the $\lambda\delta$-GdA setting the parameter h of Definition 3 to $h(k) := k + 2$. This choice ensures that the sorts $\star 0$ (Type) and $\star 1$ (Prop) do not have a common upper bound in the sort hierarchy. In our opinion, this situation is implied in CC.

Strategy	First effective use
Validation-based	`imp"l"`
Position-based (on openers)	`imp"l"`
Position-based (on constants)	`t5"l-some"`
Degree-based	`t9"l-e-st-eq-landau-n-rt-rp-r-c-v9"`

Fig. 14. Effective use of disambiguation strategies.

Step	Amount
β-contraction	907865
δ-expansion (on variables)	451799
δ-expansion (on constants)	418357

Fig. 15. Main reduction steps of the RTM.

Input	System	Execution (secs)	Task
QE-GdA	Helena 0.8.2	01.02 to 01.05	disambiguation, validation in $\lambda\delta$-3
CC-GdA	Coq 8.4.3 (no VM)	23.99 to 24.18	type checking in the refined CC
	Coq 8.4.3 (with VM)	run was halted after 60 minutes	

All systems are implemented in the Objective Caml programming language

Fig. 16. Time of one run (min. and max. on 31 runs).

Given the absence of a theoretical study, we can just conjecture that our translation of Aut-QE into $\lambda\delta$ version 3, and in turn, into the refined CC preserves correctness. The fact that Coq 8.4.3 validates the CC-GdA, supports this idea.

5. CONCLUSION AND FUTURE WORK

In [Gui09a] we describe the $\lambda\delta$-GdA: a presentation of the QE-GdA into the formal system $\lambda\delta$ version 2, experimentally equipped with *sort inclusion* to this end.

In this paper we take a step further by describing the CC-GdA: a presentation of the $\lambda\delta$-GdA in CC. The point at issue is assigning a *layer* to Automath's unified binders, *i.e.*, separating λ-abstractions and Π-abstractions. For this purpose, we replace $\lambda\delta$ version 2, a calculus having a single binder, with $\lambda\delta$ version 3 first introduced in Section 3.1: a system featuring infinite (actually, $\omega + 1$) binders, properly managing sort inclusion through υ-contraction.

In Section 2.5 and Section 3.5 we discuss the three strategies we implemented for assigning such layers to binders. The reader should note that each strategy is effective, in that it considers binders ignored by the other strategies. We show in Figure 14 the first constant of the QE-GdA on which each strategy is effective.

Our analysis reveals that some unified binders correspond to λ-abstractions and to Π-abstractions at the same time. Brown has an *ad hoc* automated procedure [Bro11] to solve this situation by formally η-expanding these inconsistent binders.

On the contrary, our approach is to apply these expansions ($\forall$-introductions, from the logical standpoint) by hand on the QE-GdA as we show in Section 3.6.

Helena 0.8.2, our processor for $\lambda\delta$ version 3 implemented in the Caml programming language, verifies the QE-GdA by operating the amount of β-contractions and of δ-expansions shown in Figure 15. The δ-expansions on variables come from the

Original presentation in the QE-GdA (Aut-QE concrete syntax)

```
+l
@[a:'prop'][b:'prop']
imp:=[x:a]b:'prop'
[a1:a][i:imp(a,b)]
mp:=<a1>i:b
a@refimp:=[x:a]x:imp(a,a)
b@[c:'prop'][i:imp(a,b)][j:imp(b,c)]
trimp:=[x:a]<<x>i>j:imp(a,c)
@con:='prim':'prop'
a@not:=imp(con):'prop'
wel:=not(not(a)):'prop'
[a1:a]
weli:=[x:not(a)]<a1>x:wel(a)
a@[w:wel(a)]
et:='prim':a
a@[c1:con]
cone:=et([x:not(a)]c1):a
```

Corresponding presentation in the CC-GdA (Coq concrete syntax)

```
Definition l_imp := (fun (a:Prop) => (fun (b:Prop) =>
                    ((forall (x:a), b) : Prop))).

Definition l_mp := (fun (a:Prop) => (fun (b:Prop) =>
                   (fun (a1:a) => (fun (i:l_imp a b) =>
                   (i a1 : b))))).

Definition l_refimp := (fun (a:Prop) =>
                       ((fun (x:a) => x) : l_imp a a)).

Definition l_trimp := (fun (a:Prop) => (fun (b:Prop) => (fun (c:Prop) =>
                      (fun (i:l_imp a b) => (fun (j:l_imp b c) =>
                      ((fun (x:a) => j (i x)) : l_imp a c)))))).

Axiom l_con : Prop.

Definition l_not := (fun (a:Prop) =>
                    (l_imp a l_con : Prop)).

Definition l_wel := (fun (a:Prop) =>
                    (l_not (l_not a) : Prop)).

Definition l_weli := (fun (a:Prop) => (fun (a1:a) =>
                     ((fun (x:l_not a) => x a1) : l_wel a))).

Axiom l_et : (forall (a:Prop), (forall (w:l_wel a), a)).

Definition l_cone := (fun (a:Prop) => (fun (c1:l_con) =>
                     (l_et a (fun (x:l_not a) => c1) : a))).
```

Fig. 17. First ten constants of the Grundlagen.

β-reductum of Figure 5(β). As a separate task, Helena 0.8.2 outputs the $\lambda\delta$-GdA in several formats including the CC-GdA for Coq 8.4.3 and, more recently, a λProlog representation. The authors of [DGST15] report that an implementation in the

abstraction	λC	CC			
unrestricted	$(\Box, \Box)$	$(\Box_T, \Box_T)$	$(\Box_T, \Box_P)$	$(\Box_P, \Box_T)$	$(\Box_P, \Box_P)$
	$(\Box, \star)$	$(\Box_T, \mathrm{Type})$	$(\Box_T, \mathrm{Prop})$	$(\Box_P, \mathrm{Type})$	$(\Box_P, \mathrm{Prop})$
	$(\star, \Box)$	$(\mathrm{Type}, \Box_T)$	$(\mathrm{Type}, \Box_P)$	$(\mathrm{Prop}, \Box_T)$	$(\mathrm{Prop}, \Box_P)$
	$(\star, \star)$	$(\mathrm{Type}, \mathrm{Type})$	$(\mathrm{Type}, \mathrm{Prop})$	$(\mathrm{Prop}, \mathrm{Type})$	$(\mathrm{Prop}, \mathrm{Prop})$
restricted	$(\star, \Box)$		$(\mathrm{Type}, \Box_P)$		$(\mathrm{Prop}, \Box_P)$
	$(\star, \star)$	$(\mathrm{Type}, \mathrm{Type})$	$(\mathrm{Type}, \mathrm{Prop})$	$(\mathrm{Prop}, \mathrm{Type})$	$(\mathrm{Prop}, \mathrm{Prop})$

Note: we assume $(\mathrm{Type} : \Box_T)$ and $(\mathrm{Prop} : \Box_P)$.

Fig. 18. The categories of abstractions in the CC-GdA.

λProlog programming language of the discussed validation procedure for $\lambda\delta$ version 3, processes this representation of the $\lambda\delta$-GdA without errors or warnings.

The whole source code of Helena 0.8.2 amounts to 350 KiB including comments.

The λProlog implementation of the mere validation procedure is 50 clauses long.

On our hardware, a 3 GHz Intel processor (1.3 MHz bus, 12 MB cache) with 10K rpm (3 Gb/s SATA) hard drives, we measured the execution times of Figure 16 concerning Helena (processing the QE-GdA), and Coq (processing the CC-GdA).

We stress that the CC-GdA is a faithful presentation of the corrected QE-GdA in that the QE-GdA is $\alpha\delta\eta$-equivalent to the CC-GdA, once abstractions and function types are replaced by the corresponding unified binding constructions.

Automath η-equivalence solves the incompatibilities between Aut-QE and CC, δ-equivalence is introduced for convenience, and α-equivalence is necessary because of different naming conventions in the QE-GdA and in Coq.

Figure 17 shows the first ten constants of the Grundlagen as they appear in the QE-GdA and in the CC-GdA. The reader sees both definitions and axioms.

Our work shows that CC is an upper-bound system for validating the CC-GdA, but we may ask whether a subsystem can be used as well. The mechanical inspection of the abstractions occurring in the $\lambda\delta$-GdA shows the situation of Figure 18, from which we argue that validation really needs the full power of CC, including the impredicative Π's of category $(\Box, \star)$ according to the classification in [Bar93].

On the other hand, if we follow Automath's perspective and we present block openers as λ^∞-abstractions, we see that the $\lambda\delta$-GdA is valid in $\Lambda_\infty + \lambda P$.

This system only allows predicative constructions [Gui09b], and is located in the middle of the diagonal connecting λP and λC in the λ-Cube [KLN01]. That is, in the center of its right face. In particular, we note that the λ^∞-abstractions of category $(\Box, \star)$ are much less expressive than the corresponding Π-abstractions.

Our translation of the QE-GdA into CC and the one of [Bro11] are similar in concept. Both need dynamic analysis to disambiguate some unified binders occurring in the text, and rely on verifying the text in a suitable type theory for this task. However, we see some important differences. Firstly, Brown's automated procedure for solving type inconsistencies applies 25 $\forall$-introductions to the text, whereas we apply just 21 $\forall$-introductions by hand. In this respect, our translated text is more faithful to the original than Brown's one. Secondly, Brown relies on Contextual Pure Type Systems, whose verification algorithm is based on type checking. On the contrary we rely on $\lambda\delta$ version 3, whose verification algorithm is theoretically faster being based on validation. Unfortunately we cannot compare the two verification algorithms at the moment because our verifier is written in Caml while

Brown's one is written in Lisp. As a minor remark, we note that Brown translates Automath definitions to Coq by duplicating their context on their type and body. On the contrary, we avoid this duplication by using type-annotated terms. Thus, our translated text is even more faithful to the original than Brown's one.

On the other hand, Brown removes the (many) unnecessary context items from the translated text, while we leave this optimization for the future. Thus, Coq 8.4.3 verifies his translated text 18% faster than ours on our hardware.

As of now, the CC-GdA is a user-level script consisting of a flat sequence of lines, each declaring or defining a constant of the QE-GdA in the syntax of CC.

In order to be usable as a background for formalized mathematics, this script must be improved by making the original structure of the Grundlagen explicit.

In particular, we would like to see definitions and propositions typeset with domain-specific mathematical notation. Proofs should appear in a domain-specific language as well, and the whole matter should be organized in different files respecting the system of chapters and sections that we see in [Lan65].

Such a step will require to operate manually of the $\lambda\delta$-GdA with the help of a dedicated technology supervising crucial aspects of the work. For instance, defining and inserting notations, or applying semantics-preserving changes.

The QE-GdA, the $\lambda\delta$-GdA and the CC-GdA, as well as Helena 0.8.2, are available at $\lambda\delta$ Web site: <`http://lambdadelta.info/implementation.html#v2`>.

ACKNOWLEDGMENTS

I wish to thank the anonymous referees for valuable suggestions that helped me to improve this text and to achieve a clearer understanding of $\lambda\delta$ version 3.

I wish to dedicate this work to K. Barr for having patiently awaited my e-mail replies while I was working on the systems of the $\lambda\delta$ family over the years.

References

[AP10] A. Abel and B. Pientka. Explicit Substitutions for Contextual Type Theory. In K. Crary and M. Miculan, editors, *Proceedings 5th International Workshop on Logical Frameworks and Meta-languages: Theory and Practice (LFMTP 2010)*, volume 34 of *EPTCS*, pages 5–20, September 2010.

[Bar93] H.P. Barendregt. Lambda Calculi with Types. *Osborne Handbooks of Logic in Computer Science*, 2:117–309, 1993.

[BKN96] R. Bloo, F. Kamareddine, and R. Nederpelt. The Barendregt Cube with definitions and generalised reduction. *Information and Computation*, 126(2):123–143, 1996.

[Bro11] C.E. Brown. Faithful Reproductions of the Automath Landau Formalization. Technical Report, 2011.

[CH88] T. Coquand and G. Huet. The Calculus of Constructions. *Information and Computation*, 76(2-3):95–120, March 1988.

[CH00] P.L. Curien and H. Herbelin. The duality of computation. In *5th ACM SIGPLAN int. conf. on Functional programming (ICFP '00)*, volume 35/9 of *ACM SIGPLAN Notices*, pages 233–243, New York, USA, Sept 2000. ACM Press.

[Chu41] A. Church. *The calculi of lambda-conversion*, volume 6 of *Annals of Mathematics Studies*. Princeton University Press, Princeton, USA, 1941.

[Coq15] Coq development team. *The Coq Proof Assistant Reference Manual: release 8.4pl6*. INRIA, Orsay, France, April 2015.

[Cos96] Y. Coscoy. A Natural Language Explanation for Formal Proofs. In C. Retoré, editor, *Int. Conf. on Logical Aspects of Computational Linguistics (LACL)*, volume 1328 of *Lecture Notes in Artificial Intelligence*, pages 149–167, Berlin, Germany, Sept 1996. Springer.

[dB78] N.G. de Bruijn. A namefree lambda calculus with facilities for internal definition of expressions and segments. TH-report 78-WSK-03, Department of Mathematics, Eindhoven University of Technology, Eindhoven, The Netherlands, August 1978.

[dB91] N.G. de Bruijn. A plea for weaker frameworks. In *Logical Frameworks*, pages 40–67. Cambridge University Press, Cambridge, UK, 1991.

[dB93] N.G. de Bruijn. Algorithmic definition of lambda-typed lambda calculus. In *Logical Environments*, pages 131–145. Cambridge University Press, Cambridge, UK, 1993.

[dB94a] N.G. de Bruijn. Generalizing Automath by Means of a Lambda-Typed Lambda Calculus. In *Selected Papers on Automath [NGdV94]*, pages 313–337. North-Holland Pub. Co., Amsterdam, The Netherlands, 1994.

[dB94b] N.G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. In *Selected Papers on Automath [NGdV94]*, pages 375–388. North-Holland Pub. Co., Amsterdam, The Netherlands, 1994.

[dB94c] N.G. de Bruijn. Some extensions of AUTOMATH: the AUT-4 family. In *Selected Papers on Automath [NGdV94]*, pages 283–288. North-Holland Pub. Co., Amsterdam, The Netherlands, 1994.

[dG91] P. de Groote. Nederpelt's calculus extended with a notion of context as a logical framework. In *Logical Frameworks*, pages 69–86. Cambridge University Press, Cambridge, UK, 1991.

[dG93] P. de Groote. Defining λ-Typed λ-Calculi by Axiomatising the Typing Relation. In *10th Annual Symposium on Theoretical Aspects of Computer Science (STACS'93)*, volume 665 of *Lecture Notes in Computer Science*, pages 712–723, Berlin, Germany, 1993. Springer.

[DGST15] C. Dunchev, F. Guidi, C. Sacerdoti Coen, and E. Tassi. ELPI: fast, Embeddable, λProlog Interpreter. In M. Davis, A. Fehnker, A. McIver, and A. Voronkov, editors, *Proceedings of 20th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-20)*, volume 9450 of *Lecture Notes in Computer Science*, pages 460–468, Berlin, Germany, December 2015. Springer.

[dV94] R.C. de Vrijer. Big trees in a λ-calculus with λ-expressions as types. In *Selected Papers on Automath [NGdV94]*, pages 469–492. North-Holland Pub. Co., Amsterdam, The Netherlands, 1994.

[Gui09a] F. Guidi. Landau's "Grundlagen der Analysis" from Automath to lambda-delta. Technical Report UBLCS 2009-16, University of Bologna, Bologna, Italy, September 2009.

[Gui09b] F. Guidi. The Formal System $\lambda\delta$. *Transactions on Computational Logic*, 11(1):5:1–5:37, online appendix 1–11, November 2009.

[Gui10] F. Guidi. An Efficient Validation Procedure for the Formal System $\lambda\delta$. In F. Ferreira, H. Guerra, E. Mayordomo, and J. Rasga, editors, *Local Proceedings of 6th Conference on Computability in Europe (CiE 2010)*, pages 204–213. Centre for Applied Mathematics and Information Technology, Department of Mathematics, University of Azores, Ponta Delgada, Portugal, July 2010.

[Gui15] F. Guidi. Extending the Applicability Condition in the Formal System $\lambda\delta$. Technical Report AMS Acta 4411, University of Bologna, Bologna, Italy, December 2015.

[KBN99] F. Kamareddine, R. Bloo, and R.P. Nederpelt. On π-conversion in the λ-cube and the combination with abbreviations. *Annals of Pure and Applied Logic*, 97(1-3):27–45, 1999.

[KLN01] F. Kamareddine, T. Laan, and R. Nederpelt. Refining the Barendregt Cube using Parameters. In H. Kuchen and K. Ueda, editors, *Functional and Logic Programming, 5th International Symposium (FLOPS 2001)*, volume 2024 of *Lecture Notes in Computer Science*, pages 375–389, Berlin, Germany, March 2001. Springer.

[KLN03] F. Kamareddine, T. Laan, and R. Nederpelt. De Bruijn's Automath and Pure Type Systems. In F. Kamareddine, editor, *Thirty Five Years of Automating Mathematics*, volume 28 of *Kluwer Applied Logic series*, pages 71–123. Kluwer Academic Publishers, Hingham, MA, USA, November 2003.

[KN94] F. Kamareddine and R. Nederpelt. A unified approach to Type Theory Through a refined λ-calculus. *Theoretical Computer Science*, 136(1):183–216, 1994. Selected papers of the Meeting on the Mathematical Foundations of Programming Semantics (MFPS '92), Part II (Oxford, 1992).

[KN96] F. Kamareddine and R.P. Nederpelt. A useful λ-notation. *Theoretical Computer Science*, 155(1):85–109, 1996.

[Kri07] J.-L. Krivine. A call-by-name lambda-calculus machine. *Higher-Order and Symbolic Computation*, 20(3):199–207, September 2007.

[Lan65] E.G.H.Y. Landau. *Grundlagen der Analysis*. Chelsea Pub. Co., New York, NY, USA, 1965.

[Luo90] Z. Luo. *An Extended Calculus of Constructions*. PhD thesis, University of Edinburgh, 1990.

[Ned79] R.P. Nederpelt. A system of lambda calculus possessing facilities for typing and abbreviating. Part I: Informal introduction. Memorandum 1979-02, Department of Mathematics, Eindhoven University of Technology, Eindhoven, The Netherlands, March 1979.

[Ned80] R.P. Nederpelt. A system of lambda calculus possessing facilities for typing and abbreviating. Part II: Formal description. Memorandum 1980-11, Department of Mathematics, Eindhoven University of Technology, Eindhoven, The Netherlands, June 1980.

[NG14] R. Nederpelt and H. Geuvers. *Type Theory and Formal Proof*. Cambridge University Press, Cambridge, UK, November 2014.

[NGdV94] R.P. Nederpelt, J.H. Geuvers, and R.C. de Vrijer, editors. *Selected Papers on Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics*, Amsterdam, The Netherlands, 1994. North-Holland Pub. Co.

[RP04] S. Ronchi Della Rocca and L. Paolini. *The Parametric Lambda Calculus*. Texts in Theoretical Computer Science, An EATCS Series. Springer, Berlin, Germany, November 2004.

[RS94] N.J. Rehof and M.H. Sørensen. The λ_Δ-calculus. In M. Hagiya and J.C. Mitchell, editors, *Theoretical Aspects of Computer Software*, volume 789 of *Lecture Notes in Computer Science*, pages 516–542. Springer, Berlin, Germany, 1994.

[vB77] L.S. van Benthem Jutting. Checking Landau's "Grundlagen" in the AUTOMATH System. Ph.D. thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 1977.

[vB79] L.S. van Benthem Jutting. *Checking Landau's "Grundlagen" in the Automath system*, volume 83 of *Mathematical Centre Tracts*. Mathematisch Centrum, Amsterdam, The Netherlands, 1979.

[vB94a] L.S. van Benthem Jutting. Description of AUT-68. In *Selected Papers on Automath [NGdV94]*, pages 251–273. North-Holland Pub. Co., Amsterdam, The Netherlands, 1994.

[vB94b] L.S. van Benthem Jutting. The language theory of λ_∞, a typed λ-calculus where terms are types. In *Selected Papers on Automath [NGdV94]*, pages 655–683. North-Holland Pub. Co., Amsterdam, The Netherlands, 1994.

[vD94a] D.T. van Daalen. A Description of Automath and Some Aspects of its Language Theory. In *Selected Papers on Automath [NGdV94]*, pages 101–126. North-Holland Pub. Co., Amsterdam, The Netherlands, 1994.

[vD94b] D.T. van Daalen. The language theory of Automath. In *Selected Papers on Automath [NGdV94]*, pages 163–200 and 303–312 and 493–653. North-Holland Pub. Co., Amsterdam, The Netherlands, 1994.

[Wie99] F. Wiedijk. A Nice and Accurate Checker for the Mathematical Language Automath. Documentation of the AUT checker, version 4.1, 1999.

[Wie02] F. Wiedijk. A new implementation of Automath. *Journal of Automated Reasoning*, 29(3-4):365–387, 2002.

[Zan94] I. Zandleven. A Verifying Program for Automath. In *Selected Papers on Automath [NGdV94]*, pages 783–804. North-Holland Pub. Co., Amsterdam, The Netherlands, 1994.