De Bruijn's AUTOMATH and Pure Type Systems

Fairouz Kamareddine* Twan Laan† Rob Nederpelt‡

Abstract

We study the position of the AUTOMATH systems within the framework of Pure Type Systems (PTSs). In [2, 22], a rough relationship has been given between AUTOMATH and PTSs. That relationship ignores three of the most important features of AUTOMATH: *definitions*, *parameters* and Π-reduction, because at the time, formulations of PTSs did not have these features. Since, PTSs have been extended with these features and in view of this, we revisit the correspondence between AUTOMATH and PTSs. This paper gives the most accurate description of AUTOMATH as a PTS so far.

1 Introduction

The AUTOMATH systems are the first examples of proof checkers, and in this way they are predecessors of modern proof checkers like Coq [20] and Nuprl [16]. The project started in 1967 by N.G. de Bruijn:

> "it was not just meant as a technical system for verification of mathematical texts, it was rather a life style with its attitudes towards understanding, developing and teaching mathematics." ([12]; see [44] p. 201)

Thus, the roots of AUTOMATH are not to be found in logic or type theory, but in mathematics and the mathematical vernacular [11]. De Bruijn had been wondering for years what a proof of a theorem in mathematics should be like, and how its correctness should be checked. The development of computers in the 60s made him wonder whether a machine could check the proof of a mathematical theorem, provided the proof was written in a very accurate way. De Bruijn developed the language AUTOMATH for this purpose. This language is not only (according to de Bruijn [10]) "*a language which we claim to be suitable for expressing very large parts of mathematics, in such a way that the correctness of the mathematical contents is guaranteed as long as the rules of grammar are obeyed*" but also "*very close to the way mathematicians have always been writing*". This is reflected in the goals of the AUTOMATH project:

> "1. The system should be able to verify entire mathematical theories.

*Mathematical and Computer Sciences, Heriot-Watt Univ., Riccarton, Edinburgh EH14 4AS, Scotland. Email: fairouz@cee.hw.ac.uk

†Weerdstede 45, 3431 LS Nieuwegein, The Netherlands. Email: twan.laan@wxs.nl

‡Mathematics and Computing Science, Eindhoven Univ. of Technology, P.O.Box 513, 5600 MB Eindhoven, the Netherlands. Email: r.p.nederpelt@tue.nl

2. The system should remain very general, tied as little as possible to any set of rules for logic and foundations of mathematics. Such basic rules should belong to material that can be presented for verification, on the same level with things like mathematical axioms that have to be explained to the reader.

3. The way mathematical material is to be presented to the system should correspond to the usual way we write mathematics. The only things to be added should be details that are usually omitted in standard mathematics."

([12]; see [44] pp. 209–210)

Goal 1 was achieved: Van Benthem Jutting [3] translated and verified Landau's "Grundlagen der Analysis" [42] in AUTOMATH and Zucker [52] formalised classical real analysis in AUTOMATH.

As for goal 2, de Bruijn used types and a propositions as types (PAT) principle[1] that was somewhat different from Curry and Howard's [17, 28]. The appearance of types in AUTOMATH finds its roots in de Bruijn's contacts with Heyting, who made de Bruijn familiar with the intuitionistic intertpretation of the logical connectives (see [26, 40]). The interpretation of the proof of an implication $A \to B$ as an algorithm to transform any proof of A into a proof of B, so in fact a function from proofs of A to proofs of B, gave rise to interpret a proposition as a class (a type) of proofs. De Bruijn who was not influenced by developments in λ-calculus or type theory when he started his work on AUTOMATH, discovered this notion of "proofs as objects", better known as "propositions as types", independently from Curry [17] and Howard [28]. Curry and Howard identified the logical implication and the universal quantifier with function types, following Heyting's intuitionistic interpretation of logical connectives. In doing so, they do not leave a possibility for a different interpretation of implication and universal quantification. Using PAT in de Bruijn's style, the rules for manipulating the logical connectives must always be made explicit by the user (for example see Sections 12 and 13 of [4]). This makes it possible to give interpretations of logical connectives that are not based on interpreting implication and universal quantification by a function type (see [41]).

De Bruijn spent a lot of effort on goal 3. He studied the language of mathematics in great depth [11] and used the following features to achieve goal 3:

- The use of books. Like a mathematical text, AUTOMATH is written line by line. Each line may refer to definitions or results given in earlier lines.
- The use of definitions and parameters. Without definitions, expressions become too long. Also, a definition gives a name to a certain expression making it easy to remember what the use of the definiens is.

As AUTOMATH was developed independently from other developments in the world of type theory and λ-calculus, and as it invented powerful typing ideas that were later adopted in influential type systems (cf. [2]), there are many things to be explained in (and learned from) the relation between the various AUTOMATH languages and other type theories. Type theory was originally invented by Bertrand Russell to exclude the paradoxes that arose from Frege's

[1] The first practical use of the propositions-as-types principle is found in AUTOMATH.

"Begriffschrift" [21]. It was presented in 1910 in the famous "Principia Mathematica" [51] and simplified by Ramsey and Hilbert and Ackermann. In 1940, Church combined his theory of functions, the λ-calculus with the simplified type theory, resulting in the influencial "simple theory of types" [15]. Since, many influential type systems have been developed. Eight of the most important such systems have been unified in the Barendregt cube [2]. Terlouw [50] and Berardi [5] extended independently Barendregt's work into a general framework leading to the so-called Pure Type Systems (PTSs [2]).

In this paper we focus on the relation between AUTOMATH and Pure Type Systems (PTSs). Both [2] and [22] mention this relation in a few lines, but as far as we know a satisfactory explanation of the relation between AUTOMATH and PTSs is not available. Moreover, both [2] and [22] consider AUTOMATH without one of its most important mechanisms: definitions and parameters. But definitions and parameters are extremely powerful in AUTOMATH. Even the AUTOMATH system PAL, which roughly consists of the definition system of AUTOMATH only, is able to express some simple mathematical reasoning (see Section 5 of [10]). According to de Bruijn [12] this is "*due to the fact that mathematicians worked with abbreviations all the time already*". Recent developments on the use of definitions and parameters in PTSs [31, 41, 32, 33, 47] justify renewed research on the relation between AUTOMATH and PTSs.

- Section 2 describes PAT, PTSs and the basic AUTOMATH system AUT-68.
- Section 3 discusses how we can transform AUT-68 into a PTS. Some properties of AUT-68 are unusual for PTSs: • η-reduction; • Π-application and Π-reduction (as AUT-68 does not distinguish λ and Π: both $\Pi x{:}A.B$ and $\lambda x{:}A.B$ are denoted by $[x{:}A]B$); • a definition system; • a parameter mechanism. We do not consider η-reduction as an essential feature of AUTOMATH, and focus on its most characteristic type-theoretical features: definitions and parameter. In systems with Π-application, Π behaves like λ, and there is a rule of Π-reduction: $(\Pi x{:}A.B)N \to_\Pi B[x{:=}N]$. We leave the features of Π-application and Π-reduction till Section 5.
- In Section 4, we give a system $\lambda 68$ that is (almost) a PTS. In $\lambda 68$, definitions play an active role. We show that $\lambda 68$ has the usual properties of PTSs and can be seen as AUT-68 without η-reduction, Π-application and Π-reduction. There is no direct parameter system in $\lambda 68$ either, but parameters are hidden in the rules for the construction of product types.
- In Section 5 we discuss how $\lambda 68$ can be extended with *direct* parameters and with Π-application and Π-reduction. We also discuss how our approach can be extended to other AUTOMATH systems like AUT-QE where the identification of λ and Π is more subtle than that of AUT-68 and it is not easy to tell whether $[x{:}A]B$ should stand for $\lambda x{:}A.B$ or $\Pi x{:}A.B$ in PTSs. In addition to AUT-QE, we reflect on $\Delta\Lambda$ (cf. [44], B.7) where terms are presented as lambda trees and to each AUTOMATH book, there corresponds a single lambda tree whose correctness is equivalent to that of the book. We conclude in Section 6.

2 AUTOMATH, PAT, PTSs and AUT-68

Basic to AUTOMATH is the PAT principle commonly known as the Curry-Howard isomorphism, although it was also invented independently by de Bruijn who applied it in a different way to that of Howard and Curry. Many other proof checkers and theorem provers, like Coq [20], Nuprl [16] and LF [23], use the PAT principle. In Section 2.1 we explain the origin of the PAT principle. Then, in Section 2.2 we introduce PTSs and we devote the rest of this section to AUTOMATH with its formulation of lines, books and definitions.

During the AUTOMATH-project, several AUTOMATH-languages have been developed. They all have two mechanisms for describing mathematics:

- The typed λ-calculus, with the important features of λ-abstraction, λ-application and β-reduction.
- The use of definitions and parameters.

The latter mechanism is the same for most AUTOMATH-systems, and the difference between the various systems is mainly caused by different λ-calculi that are included. In this section we describe the system AUT-68 [4, 9, 19] which not only is one of the first AUTOMATH-systems, but also a system with a relatively simple typed λ-calculus, which makes it easier to focus on the (less known) mechanism for definitions and parameters. A more extensive description of AUT-68 on which our description below is based, can be found in [4, 9, 19].

2.1 Propositions as Types and Proofs as Terms

Although Church's simply typed λ-calculus has logical symbols like $\vee$, $\forall$, it cannot be seen as a logical system. If one wants to make logical derivations, one has to build a logical system on top of it. Type theory nowadays plays an important role in logic in a different way: it can be used as a logical system itself. This use of type theory is generally known as "propositions as types" or "proofs as terms". As both expressions abbreviate to PAT, we will use this abbreviation to indicate both "propositions as types" and "proofs as terms". PAT only partially covers the idea of using type theory as a logical system. "Proofs as terms" already suggests an important advantage of using type theory as a logical system: here proofs are first-class citizens of the logical system, whilst for many other logical systems, proofs are rather complex objects outside the logic (for example: derivation trees), and therefore cannot be easily manipulated.

Below we mention some origins of the PAT principle.

Intuitionistic logic

The idea of PAT originates in the formulation of intuitionistic logic. Though it is not correct that "intuitionistic logic" is simply the logic that is used in intuitionistic mathematics[2], there are frequently occurring constructions in intuitionistic

[2] "Intuitionistic logic" is standard terminology for "logic without the law of the excluded middle". The terminology suggests that it is "the logic that is used in intuitionism". However,

mathematics that have a logical counterpart. One of these constructions is the proof of an implication. Heyting [25] describes the proof of an implication $a \Rightarrow b$ as: deriving a solution for the problem b from the problem a. Kolmogorov [40] is even more explicit, and describes a proof of $a \Rightarrow b$ as the construction of a method that transforms each proof of a into a proof of b. This means that a proof of $a \Rightarrow b$ can be seen as a *(constructive) function* from the proofs of a to the proofs of b. In other words, the proofs of the proposition $a \Rightarrow b$ form exactly the set of functions from the set of proofs of a to the set of proofs of b. This suggests to identify a proposition with the set of its proofs. Now *types* are used to represent these sets of proofs. An element of such a set of proofs is represented as a *term* of the corresponding type. This means that propositions are interpreted as *types*, and proofs of a proposition a as *terms of type* a.

Curry

PAT was, independently from Heyting and Kolmogorov, discovered by Curry and Feys [17]. In paragraph 8C of [17], Curry describes so-called F-objects, which correspond more or less to the simple types of Church in [15]. As a basis, a list of primitive objects $\vartheta_1, \vartheta_2, \ldots$ is chosen. All these primitive objects are F-objects. Moreover, if α and β are F-objects, then so is $\mathsf{F}\alpha\beta$. Here, F is a new symbol. $\mathsf{F}\alpha\beta$ must be interpreted as the class of functions from α to β. If α is an F-object, then the statement $\vdash \alpha X$ must be interpreted as "the object X belongs to α". The *rule*-F is adopted: if $\vdash \mathsf{F}XYZ$ and $\vdash XU$ then $\vdash Y(ZU)$. This rule immediately corresponds to the application-rule of Church's λ-calculus and says: if Z belongs to $\mathsf{F}XY$ and U belongs to X, then ZU belongs to Y.

Earlier in [17], Curry gave the implication combinator P with the *rule*-P: if $\vdash \mathsf{P}XY$ and $\vdash X$ then $\vdash Y$. $\mathsf{P}XY$ is interpreted as the proposition "if X then Y". Curry notices that rule-P has similar behaviour to rule-F.

Curry is the first to give a formalisation of PAT. For each F-object α he defines a proposition α^P by: $\vartheta_i^P \equiv \vartheta_i$ and $(\mathsf{F}\alpha\beta)^P \equiv \mathsf{P}\alpha^P\beta^P$.[3] Curry then

intuitionism (i.e., the philosophy of Brouwer and the mathematics based on it) declares mathematics to be independent of logic. According to that philosophy, a proof of a mathematical theorem is a method to read that theorem as a tautology. The fact that one needs a list of tautologies before the proof of more complicated theorems becomes clear, only indicates that the constructions we make are too complicated to be comprehended immediately. Mathematics itself however, is a construction in one's mind, independent of logic:

> "Een logische opbouw der wiskunde, onafhankelijk van de wiskundige intuïtie, is onmogelijk — daar op die manier slechts een taalgebouw wordt verkregen, dat van de eigenlijke wiskunde onherroepelijk gescheiden blijft — en bovendien een contradictio in terminis — daar een logisch systeem, zoo goed als de wiskunde zelf, de wiskundige oer-intuïtie nodig heeft"
>
> (*Over de Grondslagen der Wiskunde* [8], p. 180)

(A logical construction of mathematics, independent of the mathematical intuition, is impossible — for by this method no more is obtained than a linguistic structure, which irrevocably remains separated from mathematics — and moreover it is a contradictio in terminis — because a logical system needs the basic intuition of mathematics as much as mathematics itself needs it. [Translation from [27]]).

[3] Remark that Curry's function $\alpha \mapsto \alpha^P$ is in fact an embedding of types in propositions

shows that the types-as-propositions embedding $\alpha \mapsto \alpha^P$ is sound and complete: if $\mathsf{F}_m X_1 \cdots X_m Y$ is an abbreviation of $\mathsf{F}X_1(\mathsf{F}X_2(\ldots(\mathsf{F}X_m Y)\ldots))$ then:

> "If $\vdash \mathsf{F}_m \xi_1 \cdots \xi_m \eta X$ then $\vdash (\mathsf{F}_m \xi_1 \cdots \xi_m \eta)^P$. Moreover, if $\vdash \mathsf{F}_m \xi_1 \cdots \xi_m \eta X$ is derivable from the premises $\vdash \alpha_i a_i$ $(i = 1, \ldots, p)$ then $\vdash (\mathsf{F}_m \xi_1 \cdots \xi_m \eta)^P$ is derivable from the premises $\vdash \alpha_i^P$ $(i = 1, \ldots, p)$."
>
> ([17], paragraph 9E, Theorem 1)

> "If $\vdash (\mathsf{F}_m \xi_1 \cdots \xi_m \eta)^P$ is derivable by rule-P from the premises $\vdash \alpha_i^P$, then for each derivation of this fact and each assignment of $a_1, \ldots, a_p$ to $\alpha_1, \ldots, \alpha_p$ respectively there exists an X such that $\vdash \mathsf{F}_m \xi_1 \cdots \xi_m \eta X$ is derivable from the premises $\vdash \alpha_i a_i$ $(i = 1, \ldots, p)$ by rule-F alone."([17], paragraph 9E, Theorem 2)

The treatment of PAT in [17] is mainly directed towards Propositions as Types. Proofs as terms are implicitly present in the theory of [17]: the term X in the proof of Theorem 1 of [17] can be seen as a proof of the proposition $(\mathsf{F}_m \xi_1 \cdots \xi_m \eta)^P$. But this is not made explicit in [17].

Example 1 As an example, we show the deduction of the proposition $A \to A$ from the logical axioms $X \to Y \to X$[4] (the K-*axiom*) and $(X \to Y \to Z) \to (X \to Y) \to X \to Z$ (the S-*axiom*), both in the style of the combinator P and in the PAT-style. Both derivations correspond to the derivation of the proposition $A \to A$ in natural deduction style, with the use of modus ponens, and axioms $X \to Y \to X$ and $(X \to Y \to Z) \to (X \to Y) \to X \to Z$ only:

$$\dfrac{\dfrac{\vdash (A \to (A \to A) \to A) \to (A \to A \to A) \to A \to A \qquad \vdash A \to (A \to A) \to A}{\vdash (A \to A \to A) \to A \to A} \qquad \vdash A \to A \to A}{\vdash A \to A}.$$

- We use $\mathsf{P}_m X_1 \cdots X_m Y$ as an abbreviation for $\mathsf{P}X_1(\mathsf{P}X_2(\ldots(\mathsf{P}X_m Y)\ldots))$. So $\mathsf{P}_m X_1 \cdots X_m Y$ can be interpreted as the proposition $X_1 \to X_2 \to \cdots X_m \to Y$. In this notation, Rule-P is: $\dfrac{\vdash \mathsf{P}_{m+1} X_0 \cdots X_m Y \qquad \vdash X_0}{\vdash \mathsf{P}_m X_1 \cdots X_m Y}$.

  For terms X, Y, Z, we take the following axioms:

  (K): $\vdash \mathsf{P}_2 XYX$;

  (S): $\vdash \mathsf{P}_3(\mathsf{P}_2 XYZ)(\mathsf{P}XY)XZ$.

  Let A be a term. From the axioms we derive $\vdash \mathsf{P}AA$, using rule-P:

$$\dfrac{\dfrac{\vdash \mathsf{P}_3(\mathsf{P}_2 A(\mathsf{P}AA)A)(\mathsf{P}A(\mathsf{P}AA))AA \qquad \vdash \mathsf{P}_2 A(\mathsf{P}AA)A}{\vdash \mathsf{P}_2(\mathsf{P}A(\mathsf{P}AA))AA} \qquad \vdash \mathsf{P}A(\mathsf{P}AA)}{\vdash \mathsf{P}AA};$$

(so a types-as-propositions embedding instead of a propositions-as-types embedding).

[4] We assume that $\to$ is associative to the right, i.e. $X \to Y \to Z$ denotes $X \to (Y \to Z)$ and not $(X \to Y) \to Z$.

- In PAT-style, the situation is similar. Now we do not use any axioms, but we use some standard combinators. The combinator K (which can be compared to the λ-term $\lambda\mathtt{xy.x}$) has type F_2XYX, for arbitrary F-objects X, Y (a term can have more than one type in Curry's theory). K can be seen as a "proof" of the axiom $(\mathsf{F}_2XYX)^P$. This is indicated by putting K behind the axiom: $(\mathsf{F}_2XYX)^P\mathsf{K}$. The combinator S, comparable to the λ-term $\lambda\mathtt{xyz.xz(yz)}$, has type $\mathsf{F}_3(\mathsf{F}_2XYZ)(\mathsf{F}XY)XZ$ for arbitrary F-objects X, Y, Z. S is a "proof" of the axiom $(\mathsf{F}_3(\mathsf{F}_2XYZ)(\mathsf{F}XY)XZ)^P$. This is denoted as $(\mathsf{F}_3(\mathsf{F}_2XYZ)(\mathsf{F}XY)XZ)^P\mathsf{S}$. The derivation above now translates to:

$$\dfrac{\dfrac{\vdash \mathsf{F}_3(\mathsf{F}_2A(\mathsf{F}AA)A)(\mathsf{F}A(\mathsf{F}AA))AA\mathsf{S}}{\vdash \mathsf{F}_2A(\mathsf{F}AA)A\mathsf{K}} \qquad \vdash \mathsf{F}A(\mathsf{F}AA)\mathsf{K}}{\vdash \mathsf{F}AA(\mathsf{SKK})}.$$

  The conclusion of this derivation can be read as: SKK is a function from A to A, or, with PAT in mind: SKK is a proof of the proposition $A \to A$.

Both derivations correspond to the derivation of the proposition $A \to A$ in natural deduction style, with the use of modus ponens, and axioms $X \to Y \to X$ and $(X \to Y \to Z) \to (X \to Y) \to X \to Z$ only:

$$\dfrac{\dfrac{\vdash (A \to (A \to A) \to A) \to (A \to A \to A) \to A \to A}{\vdash A \to (A \to A) \to A} \qquad \vdash A \to A \to A}{\vdash A \to A}.$$

Howard

Howard [28] combines the argument of Curry and Feys [17] with Tait's discovery of the correspondence between cut elimination and β-reduction of λ-terms [49].

Example 2 Take this natural deduction style derivation of a proposition B:

$$\dfrac{\dfrac{\dfrac{[A]}{\boxed{\mathfrak{D}_1}}}{\dfrac{B}{A \to B}} \qquad \dfrac{\boxed{\mathfrak{D}_2}}{A}}{B}$$

Here, $[A]$ denotes that the assumption A has been discharged at the point where we concluded $A \to B$ from B. $\mathfrak{D}_1$ is a derivation with some assumptions of A, and conclusion B, whilst $\mathfrak{D}_2$ is a derivation with conclusion A. The derivation $\mathfrak{D}_2$ can be used to replace the assumptions of A in derivation $\mathfrak{D}_1$. This means that we can transform the derivation to:

$$\begin{array}{c}\boxed{\mathfrak{D}_2}\\ A\\ \boxed{\mathfrak{D}_1}\\ B\end{array}$$

where copies of $\mathfrak{D}_2$ have replaced the assumptions A in $\mathfrak{D}_1$.

We can decorate the two derivations above with λ-terms that represent proofs. This results in the following two deductions:

$$\dfrac{\dfrac{\begin{array}{c}[x{:}A]\\ \boxed{\mathfrak{D}_1}\\ T : B\end{array}}{(\lambda x{:}A.T) : (A \to B)} \qquad \begin{array}{c}\boxed{\mathfrak{D}_2}\\ S : A\end{array}}{((\lambda x{:}A.T)S) : B}$$

and

$$\begin{array}{c}\boxed{\mathfrak{D}_2}\\ S : A\\ \boxed{\mathfrak{D}_1}\\ T[x{:=}S] : B\end{array}$$

The assumption of A is represented by a variable x of type A. This is a natural idea: the variable expresses the idea "assume we have some proof of A". The derivation $\mathfrak{D}_1$ is represented by a λ-term T, in which the variable x may occur (we can use the assumption A in derivation $\mathfrak{D}_1$). Then the term $\lambda x{:}A.T$ exactly represents a proof of $A \to B$: it is a function that transforms any proof x of A into a proof T of B. As $\mathfrak{D}_2$ is a derivation of A (assume, S is a proof term of A), we can apply $\lambda x{:}A.T$ to S, obtaining a proof $(\lambda x{:}A.T)S$ of B.

Substituting the derivation $\mathfrak{D}_2$ for the assumptions of A in $\mathfrak{D}_1$ is nothing more than replacing the assumption "assume we have some proof of A" by the explicit proof S (i.e., substituting S for x). This gives a term T, where each occurrence of x has been replaced by S: the λ-term $T[x{:=}S]$. The proof transformation exactly corresponds to the β-reduction $(\lambda x{:}A.T)S \to_\beta T[x{:=}s]$.

This is the first time that proofs are treated as λ-terms. Howard doesn't call these λ-terms "proofs" but "constructions". Moreover, Howard's treatment of PAT pays attention to both Propositions as Types (following the line of Curry and Feys) and Proofs as Terms (by using λ-terms to represent proofs, thus following the interpretation of logical implication as given by Heyting).

Howard's discovery dates from 1969, but was not published until 1980.

De Bruijn

Independently of Curry and Feys and Howard, we find a variant of PAT in the first AUTOMATH system of de Bruijn (AUT-68 [44], [10]). Though de Bruijn was probably influenced by Heyting (see [12] in [44], p. 211), his ideas arose

independently from Curry, Feys and Howard This can be clearly seen in Section 2.4 of [9], where propositions as types (or better: proofs as terms) is implemented in the following way, differing from the method of Curry and Howard.

First, a constant `bool` is introduced. `bool` is a type: the type of propositions. If b is a term of type `bool` (so b is a proposition), then `true`(b) is a primitive notion of type `type`. `true`(b) represents the type of the proofs of b. So, a proof of proposition b is of type `true`(b) and not of type b (since propositions themselves are no types) With this "bool-style" implementation (as it was called by de Bruijn in [12]) in mind, it becomes clear why de Bruijn prefers the terminology "proofs as terms" to "propositions as types": in the bool-style, propositions are not represented as types. Only the class of proofs of such a proposition is represented as a type. Proofs however, are represented as terms, just as in Howard's implementation of PAT. So in the bool-style, the link between proposition and type is not as direct as the link between proof and term. The implementation of Howard (called "prop-style" by de Bruijn) does not make any distinction between a proposition and the type of its proofs.

The bool-style implementation has as advantage that one does not need a higher order lambda calculus to construct predicate logic. In relatively weak AUTOMATH systems such as AUT-68 one usually finds a "bool-style" implementation of PAT. It would be impossible to give a "prop-style" implementation in such a system as its λ-calculus is not strong enough to support it. In AUTOMATH systems with a more powerful λ-calculus we also find "prop-style" implementations. See [43] for a description of prop-style implementations in AUTOMATH.

Another advantage of the bool-style implementation is that one does not depend on a fixed interpretation of the logical connectives. One is free to define ones own logical system (and it is possible to base that system on the Brouwer-Heyting-Kolmogorov interpretation of the logical connectives. This has been one of the reasons for de Bruijn to implement PAT in a bool-style way (see [12]).

Though the bool-style implementation is not used in later AUTOMATH systems, it is still in use in the Edinburgh Logical Framework [23], and other systems[48]

2.2 Pure Type Systems

Lambda calculus was introduced by Church [13, 14], as a formalisation of the notion of function. With this formal notation he could formulate his set of postulates for the foundation of logic. Kleene and Rosser [38] showed that Church's set of postulates was inconsistent. The lambda calculus itself, however, appeared to be a very useful tool. Being a suitable framework for the formalisation of functions, it is not surprising that lambda calculus became an excellent tool for formalising the Simple Theory of Types [15]. This formalisation is at the basis of most modern type theories and especially at the basis of PTSs. In this section, we give the necessary machinery of PTSs needed for this paper.

Definition 3 Let $\mathbb{V}$ be a set of variables and $\mathbb{C}$ a set of constants (both countably infinite). The set $\mathbb{T}(\mathbb{V}, \mathbb{C})$ (or $\mathbb{T}$, if it is clear which sets $\mathbb{V}$ and $\mathbb{C}$ are used)

of typed lambda terms with variables from $\mathbb{V}$ and constants from $\mathbb{C}$ is defined by the following abstract syntax: $\mathbb{T} ::= \mathbb{V} \mid \mathbb{C} \mid \mathbb{T}\mathbb{T} \mid \lambda\mathbb{V}{:}\mathbb{T}.\mathbb{T} \mid \Pi\mathbb{V}{:}\mathbb{T}.\mathbb{T}$.

We use x, y, z, α, β as meta-variables over $\mathbb{V}$. In examples, we sometimes want to use some specific elements of $\mathbb{V}$; we use typewriter-style to denote such specific elements. So: $\mathtt{x}$ is a specific element of $\mathbb{V}$; while x is a meta-variable *over* $\mathbb{V}$. The variables $\mathtt{x}$, $\mathtt{y}$, $\mathtt{z}$ are assumed to be *distinct* elements of $\mathbb{V}$ (so $\mathtt{x} \not\equiv \mathtt{y}$ etc.), while meta-variables $x, y, z, \ldots$ may refer to variables in the object language that are syntactically equal. We use $A, B, C, M, N, \ldots, a, b, \ldots$ as meta-variables over $\mathbb{T}$. $\textsc{fv}(A)$, the set of *free variables* of A, and substitution $A[x{:=}B]$ are defined in the usual way. We use $\equiv$ to denote syntactical equality between lambda terms.

Terms that are equal up to a change of bound variables are taken to be syntactically equal. This allows the *Barendregt Convention* where bound variables are chosen to differ from free ones. Throughout, we let $\pi \in \{\lambda, \Pi\}$.

Notation 4

- We write $(\cdots((AB_1)B_2)\cdots B_n)$ as $AB_1\cdots B_n$.
- We write $\pi x_1{:}A_1.(\pi x_2{:}A_2.(\cdots(\pi x_n{:}A_n.A)\cdots))$ as $\pi\vec{x}{:}\vec{A}.B$, or $\pi_{i=1}^n x_i{:}A_i.A$.
- We write $A[x_m{:=}B_m]\cdots[x_n{:=}B_n]$ as $A[x_i{:=}B_i]_{i=m}^n$. If $m > n$ then $A[x_i{:=}B_i]_{i=m}^n$ denotes A. We also write $A[x_i{:=}B_i]_{i=1}^n$ as $A[\vec{x}{:=}\vec{B}]$.

Definition 5 (β-reduction) The relation $\to_\beta$ is given by the contraction rule $(\lambda x{:}A_1.A_2)B \to_\beta A_2[x{:=}B]$ and the usual compatibility rules. The relation $\twoheadrightarrow_\beta$ (resp. $=_\beta$) is the smallest reflexive and transitive (resp. equivalence) relation that includes $\to_\beta$. By $A \twoheadrightarrow_\beta^+ B$ we indicate that $A \twoheadrightarrow_\beta B$, but $A \not\equiv B$.

A term with no subterms of the form $(\lambda x{:}A_1.A_2)B$ is in *β-normal form*, or a *normal form* if no confusion arises. We write $A \to_\beta^{\mathrm{nf}} B$ (resp. $A \twoheadrightarrow_\beta^{\mathrm{nf}} B$) if $A \to_\beta B$ (resp. $A \twoheadrightarrow_\beta B$) and B is in β-normal form.

Definition 6

- A *specification* is a triple $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R})$, such that $\boldsymbol{S} \subseteq \mathbb{C}$, $\boldsymbol{A} \subseteq \boldsymbol{S} \times \boldsymbol{S}$ and $\boldsymbol{R} \subseteq \boldsymbol{S} \times \boldsymbol{S} \times \boldsymbol{S}$. The specification is *singly sorted* if $\boldsymbol{A}$ and $\boldsymbol{R}$ are (partial) functions from $\boldsymbol{S} \to \boldsymbol{S}$ and $\boldsymbol{S} \times \boldsymbol{S} \to \boldsymbol{S}$ resp. We call $\boldsymbol{S}$ the set of *sorts*, $\boldsymbol{A}$ the set of *axioms*, and $\boldsymbol{R}$ the set of (Π-formation) *rules*.
- A *context* is a finite (possibly empty) list $x_1{:}A_1, \ldots, x_n{:}A_n$ (or $\vec{x}{:}\vec{A}$) of variable declarations. $\{x_1, \ldots, x_n\}$ is the *domain* $\textsc{dom}\left(\vec{x}{:}\vec{A}\right)$ of the context. The *empty context* is denoted $\langle\rangle$. We use Γ, Δ to range over contexts.
- We extend substitutions to contexts by: $\langle\rangle[x{:=}A] \equiv \langle\rangle$; and $(\Gamma', y{:}B)[x{:=}A] \equiv \begin{cases} \Gamma'[x{:=}A] & \text{if } x \equiv y; \\ \Gamma'[x{:=}A], y{:}B[x{:=}A] & \text{if } x \not\equiv y. \end{cases}$

Though PTSs were not introduced before 1988 [5, 50] many rules are highly influenced by rules of known type systems like Church's Simple Theory of Types [15] and Automath (see 5.5.4. of [18], and Section 2).

$$\text{(axiom)}\quad \frac{}{\langle\rangle \vdash s_1 : s_2} \qquad (s_1, s_2) \in \boldsymbol{A}$$

$$\text{(start)}\quad \frac{\Gamma \vdash A : s}{\Gamma, x{:}A \vdash x : A} \qquad x \notin \mathrm{DOM}(\Gamma)$$

$$\text{(weak)}\quad \frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s}{\Gamma, x{:}C \vdash A : B} \qquad x \notin \mathrm{DOM}(\Gamma)$$

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

$$(\lambda)\quad \frac{\Gamma, x{:}A \vdash b : B \qquad \Gamma \vdash (\Pi x{:}A.B) : s}{\Gamma \vdash (\lambda x{:}A.b) : (\Pi x{:}A.B)}$$

$$\text{(appl)}\quad \frac{\Gamma \vdash F : (\Pi x{:}A.B) \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x{:=}a]}$$

$$\text{(conv)}\quad \frac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : s \qquad B =_\beta B'}{\Gamma \vdash A : B'}$$

Figure 1: Typing rules of PTSs

$\lambda{\to}$	$(*,*)$			
$\lambda 2$	$(*,*)$	$(\Box,*)$		
$\lambda \mathrm{P}$	$(*,*)$		$(*,\Box)$	
$\lambda\underline{\omega}$	$(*,*)$			$(\Box,\Box)$
$\lambda \mathrm{P2}$	$(*,*)$	$(\Box,*)$	$(*,\Box)$	
$\lambda\omega$	$(*,*)$	$(\Box,*)$		$(\Box,\Box)$
$\lambda \mathrm{P}\underline{\omega}$	$(*,*)$		$(*,\Box)$	$(\Box,\Box)$
$\lambda \mathrm{C}$	$(*,*)$	$(\Box,*)$	$(*,\Box)$	$(\Box,\Box)$

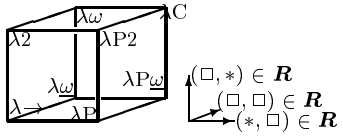

Figure 2: The Barendregt Cube

Definition 7 (Pure Type Systems) Let $\mathfrak{S} = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R})$ be a specification. The Pure Type System $\lambda\mathfrak{S}$ describes the judgements (given in Figure 7) $\Gamma \vdash_{\mathfrak{S}} A : B$ (or $\Gamma \vdash A : B$, if it is clear which $\mathfrak{S}$ is used). $\Gamma \vdash A : B$ states that A has type B in context Γ. A context Γ is *legal* if there are A, B such that $\Gamma \vdash A : B$. A term A is *legal* if there are Γ, B such that $\Gamma \vdash A : B$ or $\Gamma \vdash B : A$.

An important class of PTSs is given as eight PTSs in the Barendregt Cube [2] of Figure 2. These systems all have $\{*, \Box\}$ as set of sorts, and $*{:}\Box$ as only axiom, but differ on the Π-formation rules. We write (s_1, s_2, s_2) as (s_1, s_2).

2.3 Books, lines and expressions of AUTOMATH

In AUTOMATH, a mathematical text is thought of as being a series of consecutive "clauses". Each clause is expressed as a *line*. Lines are stored in so-called *books*. For writing lines and books in AUT-68 we need: • The symbol `type`; • A set $\mathcal{V}$ of variables; • A set $\mathcal{C}$ of constants; • The symbols ( ) [ ] : — , .

We assume $\mathcal{V}$ and $\mathcal{C}$ are infinite, $\mathcal{V} \cap \mathcal{C} = \varnothing$ and $\mathtt{type} \notin \mathcal{V} \cup \mathcal{C}$. The elements of $\mathcal{V}$ are called *block openers*, those of $\mathcal{V} \cup \mathcal{C}$ are called *identifiers* in [10].

Definition 8 (Expressions) We define the set $\mathcal{E}$ of AUT-*68-expressions* (or *expressions*) inductively as below. Sometimes we use the set $\mathcal{E}^+ \stackrel{\text{def}}{=} \mathcal{E} \cup \{\mathtt{type}\}$.

(variable) If $x \in \mathcal{V}$ then $x \in \mathcal{E}$; We use the same meta-variables and specific elements as for $\mathbb{V}$.

(parameter) If $a \in \mathcal{C}$, $n \in \mathbb{N}$ $(n \geq 0)$ and $\Sigma_1, \ldots, \Sigma_n \in \mathcal{E}$ then $a(\Sigma_1, \ldots, \Sigma_n) \in \mathcal{E}$. We call $\Sigma_1, \ldots, \Sigma_n$ the *parameters* of $a(\Sigma_1, \ldots, \Sigma_n)$;

(abstraction) If $x \in \mathcal{V}$, $\Sigma \in \mathcal{E} \cup \{\mathtt{type}\}$ and $\Omega \in \mathcal{E}$ then $[x{:}\Sigma]\Omega \in \mathcal{E}$;

(application) If $\Sigma_1, \Sigma_2 \in \mathcal{E}$ then $\langle \Sigma_2 \rangle \Sigma_1 \in \mathcal{E}$.

Remark 9

- The AUT-68-expression $[x{:}\Sigma]\Omega$ is AUTOMATH-notation for abstraction. In PTS-notation one writes $\lambda x{:}\Sigma.\Omega$ or $\Pi x{:}\Sigma.\Omega$. In a relatively simple AUTOMATH-system like AUT-68, it is easy to determine whether $\lambda x{:}\Sigma.\Omega$ or $\Pi x{:}\Sigma.\Omega$ is the correct interpretation for $[x{:}\Sigma]\Omega$. This is harder in AUTOMATH-systems with a more complex λ-calculus, like AUT-QE.
- The AUT-68-expression $\langle \Sigma_2 \rangle \Sigma_1$ is AUTOMATH-notation for the intended application of the "function" Σ_1 to the "argument" Σ_2. In PTS-notation: $\Sigma_1 \Sigma_2$. (Note the unusual *order* of "function" Σ_1 and "argument" Σ_2). The advantages of writing $\langle \Sigma_2 \rangle \Sigma_1$ instead of the classical $\Sigma_1 \Sigma_2$ are extensively discussed in [37]. In particular, if Σ_1 is a function $[x{:}\Omega_1]\Omega_2$, then $\langle \Sigma_2 \rangle \Sigma_1 \equiv \langle \Sigma_2 \rangle [x{:}\Omega_1]\Omega_2$. The argument Σ_2 and the abstraction $[x{:}\Omega_1]$ belong together: as soon as the intended application of the function Σ_1 to its argument is carried out, Σ_2 is substituted for x everywhere in Ω_2. It is convenient to put expressions that belong together next to each other. In classical notation, one writes $([x{:}\Omega_1]\Omega_2)\Sigma_2$, where Σ_2 and $[x{:}\Omega_1]$ are separated from each other by the expression Ω_2. This makes the structure of the expression less clear, in particular if Ω_2 is a very long expression.

We define $\text{FV}(A)$ in the same way as for PTSs where also $\text{FV}(a(\Sigma_1, \ldots, \Sigma_n)) \stackrel{\text{def}}{=} \bigcup_{i=1}^{n} \text{FV}(\Sigma_i)$. We adhere to the usual convention that names of bound variables in an expression differ from the free variables in that expression. We use $\equiv$ to denote syntactical equivalence (up to renaming of bound variables).

Definition 10 If $\Omega, \Sigma_1, \ldots, \Sigma_n$ are expressions (in $\mathcal{E}$), and $x_1, \ldots, x_n$ are distinct variables, then $\Omega[x_1, \ldots, x_n{:=}\Sigma_1, \ldots, \Sigma_n]$ denotes the expression Ω in which all free occurrences of $x_1, \ldots, x_n$ have simultaneously been replaced by $\Sigma_1, \ldots, \Sigma_n$. This is an expression in $\mathcal{E}$ (this can be proved by induction on the structure of Ω). Moreover, $\mathtt{type}[x_1, \ldots, x_n{:=}\Sigma_1, \ldots, \Sigma_n]$ is defined as $\mathtt{type}$.

Definition 11 (Books and lines) An AUT-68-*book* (or *book*) is a finite list (possibly empty) of (AUT-68)-lines (to be defined next). If $l_1, \ldots, l_n$ are the lines of book $\mathfrak{B}$, we write $\mathfrak{B} \equiv l_1, \ldots, l_n$. (See Example 13.)
An AUT-68-*line* (*line* if no confusion arises) is a 4-tuple $(\Gamma; k; \Sigma_1; \Sigma_2)$. Here,

- Γ is a context, i.e. a finite (possibly empty) list $x_1{:}\alpha_1, \ldots, x_n{:}\alpha_n$, where the x_is are different elements of $\mathcal{V}$ and the α_is are elements of $\mathcal{E} \cup \{\texttt{type}\}$;
- Σ_1 can be (only): $\circ$ The symbol — (if $k \in \mathcal{V}$); $\circ$ The symbol PN (if $k \in \mathcal{C}$) (PN stands for "primitive notion"); $\circ$ An element of $\mathcal{E}$ (if $k \in \mathcal{C}$);
- k is an element of $\mathcal{V} \cup \mathcal{C}$; and Σ_2 is an element of $\mathcal{E} \cup \{\texttt{type}\}$.

Remark 12 There are three sorts of Automath-lines (see Example 13):

1. $(\Gamma; k; —; \Sigma_2)$ with $k \in \mathcal{V}$. This is a *variable declaration* of the variable k having type Σ_2. This does not really add a new statement to the book, but these declarations are needed to form contexts.

   Variables can play two roles. First of all they can represent an unspecified object of a certain type (compare this to the mathematical way of speaking: "let x be a natural number"). Secondly, a variable can act as a logical assumption. This happens if the variable has as type the proof of a certain proposition A. The usual mathematical way of speaking in such a situation is not "let x be a proof of A", but: "assume A";

2. $(\Gamma; k; \text{PN}; \Sigma_2)$ with $k \in \mathcal{C}$. This line introduces a *primitive notion*: a constant k of type Σ_2. This constant can act as a primitive notion (for instance introducing the type of natural numbers, or introducing the number 0), or as an axiom. The introduction of k is *parametrised* by the context Γ. For instance, if we want to introduce the primitive notion of "logical conjunction", we do not want to have a separate primitive notion for each possible conjunction $\texttt{and}(A, B)$.[5] Instead, we want to have one primitive notion `and`, to which we can add two propositions A and B as parameters when we want to form the proposition $\texttt{and}(A, B)$. Therefore, we introduce `and` in a context $\Gamma \equiv \texttt{x:prop}, \texttt{y:prop}$. Given certain propositions A, B this enables us to form the AUT-68-expression $\texttt{and}(A, B)$;

3. $(\Gamma; k; \Sigma_1; \Sigma_2)$ with $k \in \mathcal{C}$ and $\Sigma_1 \in \mathcal{E}$. This line introduces a *definition*. The *definiendum* k is defined by the *definiens* Σ_1 and has *type* Σ_2. Definitions are parametrised like primitive notions. They help to:

   - abbreviate long expressions, clarify the book structure, and make expression manipulations efficient;
   - give a name to an expression. For instance, we can abbreviate $\texttt{S(S(S(S(S(S(S(0)))))))}$ by `7`.

Example 13 In Figure 3 we give an example of an AUTOMATH-book that introduces some elementary notions of propositional logic. We have numbered each line in the example, and use these line numbers for reference in our comments below. To keep things clear, we have omitted the types of the variables in the context. The book consists of three parts:

[5] Unlike the habit in mathematics to use only one character (possibly indexed) for a variable, AUTOMATH adopts the convention of computer science to use variables containing more than one character. So `and` represents only one variable, and not the application of `a` to `n` and `d`.

∅	prop	PN	type	(1)
∅	x	—	prop	(2)
x	y	—	prop	(3)
x,y	and	PN	prop	(4)
x	proof	PN	type	(5)
x,y	px	—	proof(x)	(6)
x,y,px	py	—	proof(y)	(7)
x,y,px,py	and-I	PN	proof(and)	(8)
x,y	pxy	—	proof(and)	(9)
x,y,pxy	and-O1	PN	proof(x)	(10)
x,y,pxy	and-O2	PN	proof(y)	(11)
x	prx	—	proof(x)	(12)
x,prx	and-R	and-I(x,x,prx,prx)	proof(and(x,x))	(13)
x,y,pxy	and-S	and-I(y,x,and-O2,and-O1)	proof(and(y,x))	(14)

Figure 3: Example of an AUTOMATH-book

- In lines 1–5 we introduce some basic material:

  1. We take the type `prop` as a primitive notion. This type can be interpreted as the type of propositions;
  2. We declare a variable `x` of type `prop`. This variable will be used in the sequel of the book;
  3. We similarly define a variable `y` of type `prop` within the context `x:prop`. For reasons of space, we do not explicitly mention the type of `x` in the context; if necessary we can find that type in line 2;
  4. Given propositions `x` and `y`, we introduce a new primitive notion, the conjunction `and(x,y)` of `x` and `y`;
  5. Given a proposition `x` we introduce the type `proof(x)` of the proofs of `x` as a primitive notion. In this way, we can use the PAT principle à la de Bruijn (cf. Section 2.1);

- In lines 6–11 we show how we can construct proofs of propositions of the form `and(x,y)`, and how we can use proofs of such propositions:

  6. Given propositions `x` and `y`, we assume that we have a `px` $\in \mathcal{V}$ of type `proof(x)`. I.e., the variable `px` represents a proof of `x`;
  7. We also assume a proof `py` of `y`;
  8. Given propositions `x` and `y`, and proofs `px` and `py` of `x` and `y`, we want to conclude that `and(x,y)` holds. This is a natural deduction axiom which we call `and-I` (and-introduction). `and-I(x,y,px,py)` is a proof of `and(x,y)`, so of type `proof(and(x,y))`.

     In line 8, we see `proof(and)` instead of `proof(and(x,y))` as the type of `and-I`. This is usual in Automath, and keeps lines short. This "default mechanism" works as follows. As the context of line 4 has two variables `x` and `y`, we conclude that `and` should always carry two parameters. In the expression `proof(and)` in line 8, no parameters

are provided for `and`. It is then assumed that the first two variables of the context of line 8 are used as "default parameters". The first two variables of the context of line 8 are `x` and `y`. Therefore, `proof(and)` in line 8 should be read as `proof(and(x,y))`.

Similarly, we can write `proof` instead of `proof(x)` in line 6. From line 5 (where `proof` is introduced) we find that `proof` carries one parameter. Writing just `proof` in line 6 means that we must use the first variable of the context of line 6, `x`, as a default parameter. We must write `proof(y)` in line 7 because `proof` would give `proof(x)`;

9. To express how we can use a proof of `and(x,y)`, first we introduce a variable `pxy` that represents an arbitrary proof of `and(x,y)`;
10. As we want `x` to hold when `and(x,y)` holds, we introduce an axiom `and-O1` (and-out, first and-elimination). Given propositions `x,y` and a proof `pxy` of `and(x,y)`, `and-O1(x,y,pxy)` is a proof of `x`;
11. Similarly, we introduce an axiom `and-O2` representing a proof of `y`;

- We can now derive some elementary theorems:

12. We want to derive `and(x,x)` from `x`. That is: from a proof of `x`, we can construct a proof of `and(x,x)`. In line 6, we introduced a variable `px` for a proof of `x`. However, we declared `px` in the context `x,y`. As we do not want a second proposition `y` to occur in this theorem, we declare a new proof variable `prx`, in the context `x`;
13. We derive our theorem: the reflexivity of the logical conjunction. Given a proposition `x`, and a proof `prx` of `x`, we can use the axiom `and-I` to find a proof of `and(x,x)`: we can use `and-I(x,x,px,px)` thanks to line 8. We give a name to this proof: `and-R`. If, anywhere in the sequel of the book, Σ is a proposition, and Ω is a proof of Σ, we can write `and-R`(Σ, Ω) for a proof of `and`(Σ, Σ). This is shorter, and more expressive, than the original expression `and-I`$(\Sigma, \Sigma, \Omega, \Omega)$;
14. We also show that `and` is symmetric: whenever `and(x,y)` holds, we also have `and(y,x)`. The idea is as follows. Given propositions `x,y` and a proof `pxy` of `and(x,y)`, we can form proofs `and-O1(x,y,pxy)` of `x` and `and-O2(x,y,pxy)` of `y`. We can feed these proofs "in reverse order" to the axiom `and-I`: the expression `and-I(y,x,and-O2,and-O1)` represents a proof of `and(y,x)`. The expression `and-O2` should be read as `and-O2(x,y,pxy)` due to the "default parameter" mechanism. Similarly, `and-O1` must be read as `and-O1(x,y,pxy)`.

2.4 Correct books

Not all books are good books. If $(\Gamma; k; \Sigma_1; \Sigma_2)$ is a line of a book $\mathfrak{B}$, the expressions Σ_1 and Σ_2 (as long as Σ_1 is not PN or —, and Σ_2 is not `type`) must be well-defined, i.e. the elements of $\mathcal{V} \cup \mathcal{C}$ occurring in them must have been established (as variables, primitive notions, or defined constants) in previous

parts of $\mathfrak{B}$. The same holds for the type assignments $x_i{:}\alpha_i$ that occur in Γ. Moreover, if Σ_1 is not PN or —, then Σ_1 must be of the same type as k, hence Σ_1 must be of type Σ_2 (within the context Γ). Finally, there should be only one definition of any object in a book, so k should not occur in the preceding lines of the book. Hence we need notions of correctness and of typing.

We write $\mathfrak{B}; \varnothing \vdash \text{OK}$ to indicate that a book $\mathfrak{B}$ is correct, and $\mathfrak{B}; \Gamma \vdash \text{OK}$ to indicate that the context Γ is correct with respect to the (correct) book $\mathfrak{B}$.[6] We write $\mathfrak{B}; \Gamma \vdash \Sigma_1 : \Sigma_2$ to indicate that Σ_1 is a correct expression of type Σ_2 (or simply a correct expression) with respect to $\mathfrak{B}$ and Γ. We also say: $\Sigma_1 : \Sigma_2$ is a correct *statement* with respect to $\mathfrak{B}$ and Γ. We write $\vdash_{\text{AUT}-68}$ if a confusion of systems arises. The following two interrelated definitions are based on [19].

Definition 14 (Correct books and contexts) A book $\mathfrak{B}$ and a context Γ are *correct* if $\mathfrak{B}; \Gamma \vdash \text{OK}$ can be derived with the rules below ($=_{\beta\text{d}}$ is given in Section 2.5. The rules use *correct statements* of Definition 15):

(axiom) $$\varnothing; \varnothing \vdash \text{OK}$$

(context ext.) $$\frac{\mathfrak{B}_1, (\Gamma; x; —; \alpha), \mathfrak{B}_2; \Gamma \vdash \text{OK}}{\mathfrak{B}_1, (\Gamma; x; —; \alpha), \mathfrak{B}_2; \Gamma, x{:}\alpha \vdash \text{OK}}$$

(book ext.: var1) $$\frac{\mathfrak{B}; \Gamma \vdash \text{OK}}{\mathfrak{B}, (\Gamma; x; —; \texttt{type}); \varnothing \vdash \text{OK}}$$

(book ext.: var2) $$\frac{\mathfrak{B}; \Gamma \vdash \Sigma_2 : \texttt{type}}{\mathfrak{B}, (\Gamma; x; —; \Sigma_2); \varnothing \vdash \text{OK}}$$

(book ext.: pn1) $$\frac{\mathfrak{B}; \Gamma \vdash \text{OK}}{\mathfrak{B}, (\Gamma; k; \text{PN}; \texttt{type}); \varnothing \vdash \text{OK}}$$

(book ext.: pn2) $$\frac{\mathfrak{B}; \Gamma \vdash \Sigma_2 : \texttt{type}}{\mathfrak{B}, (\Gamma; k; \text{PN}; \Sigma_2); \varnothing \vdash \text{OK}}$$

(book ext.: def1) $$\frac{\mathfrak{B}; \Gamma \vdash \Sigma_1 : \texttt{type}}{\mathfrak{B}, (\Gamma; k; \Sigma_1; \texttt{type}); \varnothing \vdash \text{OK}}$$

(book ext.: def2) $$\frac{\mathfrak{B}; \Gamma \vdash \Sigma_2 : \texttt{type} \qquad \mathfrak{B}; \Gamma \vdash \Sigma_1 : \Sigma_2' \;\; \mathfrak{B}; \Gamma \vdash \Sigma_2 =_{\beta\text{d}} \Sigma_2'}{\mathfrak{B}, (\Gamma; k; \Sigma_1; \Sigma_2); \varnothing \vdash \text{OK}}$$

For the (book ext.) rules, we assume $x \in \mathcal{V}$ and $k \in \mathcal{C}$ do not occur in $\mathfrak{B}$ or Γ.

Definition 15 (Correct statements) A statement $\mathfrak{B}; \Gamma \vdash \Sigma : \Omega$ is *correct* if it can be derived with the rules below (the start rule uses the notions of correct context and correct book as given in Definition 14).

[6] As the empty context will be correct with respect to any correct book.

(start) $$\frac{\mathfrak{B};\Gamma_1,x{:}\alpha,\Gamma_2 \vdash \text{OK}}{\mathfrak{B};\Gamma_1,x{:}\alpha,\Gamma_2 \vdash x{:}\alpha}$$

(parameters) $$\frac{\mathfrak{B} \equiv \mathfrak{B}_1,(x_1{:}\alpha_1,\ldots,x_n{:}\alpha_n;b;\Omega_1;\Omega_2),\mathfrak{B}_2 \quad \mathfrak{B};\Gamma \vdash \Sigma_i{:}\alpha_i[x_1,\ldots,x_{i-1}{:=}\Sigma_1,\ldots,\Sigma_{i-1}](i=1,\ldots,n)}{\mathfrak{B};\Gamma \vdash b(\Sigma_1,\ldots,\Sigma_n):\Omega_2[x_1,\ldots,x_n{:=}\Sigma_1,\ldots,\Sigma_n]}$$

(abstr.1) $$\frac{\mathfrak{B};\Gamma \vdash \Sigma_1{:}\mathtt{type} \quad \mathfrak{B};\Gamma,x{:}\Sigma_1 \vdash \Omega_1{:}\mathtt{type}}{\mathfrak{B};\Gamma \vdash [x{:}\Sigma_1]\Omega_1 : \mathtt{type}}$$

(abstr.2) $$\frac{\mathfrak{B};\Gamma \vdash \Sigma_1{:}\mathtt{type} \quad \mathfrak{B};\Gamma,x{:}\Sigma_1 \vdash \Omega_1{:}\mathtt{type} \quad \mathfrak{B};\Gamma,x{:}\Sigma_1 \vdash \Sigma_2{:}\Omega_1}{\mathfrak{B};\Gamma \vdash [x{:}\Sigma_1]\Sigma_2 : [x{:}\Sigma_1]\Omega_1}$$

(application) $$\frac{\mathfrak{B};\Gamma \vdash \Sigma_1 : [x{:}\Omega_1]\Omega_2 \quad \mathfrak{B};\Gamma \vdash \Sigma_2 : \Omega_1}{\mathfrak{B};\Gamma \vdash \langle\Sigma_2\rangle\Sigma_1 : \Omega_2[x{:=}\Sigma_2]}$$

(conversion) $$\frac{\mathfrak{B};\Gamma \vdash \Sigma : \Omega_1 \quad \mathfrak{B};\Gamma \vdash \Omega_2{:}\mathtt{type} \quad \mathfrak{B};\Gamma \vdash \Omega_1 =_{\beta\mathrm{d}} \Omega_2}{\mathfrak{B};\Gamma \vdash \Sigma : \Omega_2}$$

When using the parameter rule, we assume that $\mathfrak{B};\Gamma \vdash \text{OK}$, even if $n = 0$.

Lemma 16 *The book of Example 13 (see Figure 3) is correct.*

PROOF: We prove this for the first four lines (we leave lines 5–14 for the reader). We write (m–n) to denote the book that consists of lines m to n of Example 13.

1. By (axiom), $\varnothing;\varnothing \vdash \text{OK}$, so $(\varnothing;\mathtt{prop};\text{PN};\mathtt{type});\varnothing \vdash \text{OK}$ (book ext.: pn1).
2. By (parameters), (1–1); $\varnothing \vdash \mathtt{prop} : \mathtt{type}$. Therefore by (book ext.: var1), we have: (1–1), $(\varnothing,\mathtt{x},\text{—},\mathtt{prop});\varnothing \vdash \text{OK}$.
3. By (context ext.), (1–2); $\mathtt{x}{:}\mathtt{prop} \vdash \text{OK}$.
   Therefore by (book ext.: var1), we have: (1–2), $(\mathtt{x}{:}\mathtt{prop};\mathtt{y};\text{—};\mathtt{prop}) \vdash \text{OK}$.
4. By two applications of (context ext.), (1–3); $\mathtt{x}{:}\mathtt{prop},\mathtt{y}{:}\mathtt{prop} \vdash \text{OK}$.
   By (parameters), we have: (1–3); $\mathtt{x}{:}\mathtt{prop},\mathtt{y}{:}\mathtt{prop} \vdash \mathtt{prop}{:}\mathtt{type}$.
   Therefore by (book ext.: pn2), we have: (1–4); $\varnothing \vdash \text{OK}$. ⊠

2.5 Definitional equality

We need to describe the relation $=_{\beta\mathrm{d}}$ ("definitional equality"). This notion is based on the mechanisms of definition and abstraction/application of AUT-68. The abstraction/application mechanism provides the well-known notion of β-equality, originating from $\langle\Sigma\rangle[x{:}\Omega_2]\Omega_1 \to_\beta \Omega_1[x{:=}\Sigma]$. We need to describe the definition mechanism of AUT-68 via the notion of d-*equality*.[7]

Definition 17 (d-equality) Assume, $\mathfrak{B};\Gamma \vdash \Sigma : \Sigma'$. We define the d-*normal form* $\mathrm{nf_d}(\Sigma)$ of Σ with respect to $\mathfrak{B}$ by induction on the length of $\mathfrak{B}$. Assume $\mathrm{nf_d}(\Sigma)$ has been defined for all $\mathfrak{B}'$ with less lines than $\mathfrak{B}$, and all Σ that are correct with respect to $\mathfrak{B}'$ and a context Γ. By induction on the structure of Σ:

[7] This definition depends on the definition of derivability $\vdash$ which in turn depends on the definition of $=_{\beta\mathrm{d}}$. The definitions of correct book, correct line, correct context, correct expression and $=_{\beta\mathrm{d}}$ should be given within one definition, using induction on the length of the book. This would lead to a correct but very long definition, and that is the reason why the definitions are split into smaller parts (in this paper as well as in [19]).

- If Σ is a variable x, then $\mathrm{nf_d}(\Sigma) \stackrel{\mathrm{def}}{=} x$;
- If $\Sigma \equiv b(\Omega_1, \ldots, \Omega_n)$ and the normal forms of the Ω_is have been defined, determine a line $(\Delta; b; \Xi_1; \Xi_2)$ in the book $\mathfrak{B}$ (there is exactly one such line, and it is determined by b). Write $\Delta \equiv x_1{:}\alpha_1, \ldots, x_n{:}\alpha_n$. Distinguish:
  - $\Xi_1 \equiv —$. This case doesn't occur, as $b \in \mathcal{C}$;
  - $\Xi_1 \equiv \textsc{pn}$. Then define $\mathrm{nf_d}(\Sigma) \stackrel{\mathrm{def}}{=} b(\mathrm{nf_d}(\Omega_1), \ldots, \mathrm{nf_d}(\Omega_n))$;
  - Ξ_1 is an expression. Then Ξ_1 is correct with respect to $\mathfrak{B}'$ that contains less lines than $\mathfrak{B}$ ($\mathfrak{B}'$ doesn't contain the line $(\Delta; b; \Xi_1; \Xi_2)$, and all lines of $\mathfrak{B}'$ are lines of $\mathfrak{B}$), hence we can assume $\mathrm{nf_d}(\Xi_1)$ has been defined. Define $\mathrm{nf_d}(\Sigma) \stackrel{\mathrm{def}}{=} \mathrm{nf_d}(\Xi_1)[x_1, \ldots, x_n{:=}\mathrm{nf_d}(\Omega_1), \ldots, \mathrm{nf_d}(\Omega_n)]$;
- If $\Sigma \equiv [x{:}\Omega_1]\Omega_2$ then $\mathrm{nf_d}(\Sigma) \stackrel{\mathrm{def}}{=} [x{:}\mathrm{nf_d}(\Omega_1)]\mathrm{nf_d}(\Omega_2)$;
- If $\Sigma \equiv \langle\Omega_2\rangle\Omega_1$ then $\mathrm{nf_d}(\Sigma) \stackrel{\mathrm{def}}{=} \langle\mathrm{nf_d}(\Omega_2)\rangle\mathrm{nf_d}(\Omega_1)$.

Write $\Sigma_1 =_\mathrm{d} \Sigma_2$ if $\mathrm{nf_d}(\Sigma_1) \equiv \mathrm{nf_d}(\Sigma_2)$[8] and $=_{\beta\mathrm{d}}$ for the smallest equivalence relation containing $=_\beta$ and $=_\mathrm{d}$.

Definition 18 Σ_1 and Σ_2 are called *definitionally equal* (with respect to a book $\mathfrak{B}$) if $\Sigma_1 =_{\beta\mathrm{d}} \Sigma_2$.[9]

Instead of Definition 17, we can define d-equality via a reduction relation.

Definition 19 (δ-reduction) Let $\mathfrak{B}$ be a book, Γ a correct context with respect to $\mathfrak{B}$, and Σ a correct expression with respect to $\mathfrak{B}; \Gamma$. We define $\Sigma \to_\delta \Omega$ by the usual compatibility rules, and

(δ) If $\Sigma = b(\Sigma_1, \ldots, \Sigma_n)$, and $\mathfrak{B}$ contains a line $(x_1{:}\alpha_1, \ldots, x_n{:}\alpha_n; b; \Xi_1; \Xi_2)$ where $\Xi_1 \in \mathcal{E}$, then $\Sigma \to_\delta \Xi_1[x_1, \ldots, x_n{:=}\Sigma_1, \ldots, \Sigma_n]$.

Σ is in δ-normal form if for no expression Ω, $\Sigma \to_\delta \Omega$. We define $\twoheadrightarrow_\delta$, $\twoheadrightarrow_\delta^+$ and $=_\delta$ as usual. Again, $\to_\delta$ depends on $\mathfrak{B}$, but we drop $\mathfrak{B}$ if no confusion occurs.

Lemma 20 *1. (Church-Rosser) If $A_1 =_\delta A_2$ then there is B such that $A_1 \to_\delta B$ and $A_2 \to_\delta B$;*

2. $\mathrm{nf_d}(\Sigma)$ is the unique δ-normal form of Σ;

3. $\Sigma =_\delta \Omega$ if and only if $\Sigma =_\mathrm{d} \Omega$.

4. $\to_\delta$ is strongly normalising.

[8] Note that the d-normal form $\mathrm{nf_d}(\Sigma)$ of a correct expression Σ depends on the book $\mathfrak{B}$, and to be completely correct we should write $\mathrm{nf_{d\mathfrak{B}}}(\Sigma)$ instead of $\mathrm{nf_d}(\Sigma)$. We will, however, omit the subscript $\mathfrak{B}$ as long as no confusion arises.

[9] Definitional equality of expressions Σ_1 and Σ_2 depends on the book $\mathfrak{B}$, so we should write $=_{\beta\mathrm{d}\mathfrak{B}}$ instead of $=_{\beta\mathrm{d}}$. As before, we leave out the subscript $\mathfrak{B}$ as long as no confusion arises.

PROOF:

1. AUT-68 with $\rightarrow_\delta$ is an orthogonal term rewrite system (see [39]). Such a term rewrite system has the Church-Rosser property (see [39]);
2. It is not hard to show that $\Sigma \twoheadrightarrow_\delta \mathrm{nf_d}(\Sigma)$. By induction on the definition of $\mathrm{nf_d}(\Sigma)$ one shows that $\mathrm{nf_d}(\Sigma)$ is in δ-normal form. The uniqueness of this normal form follows from the Church-Rosser property;
3. If $\Sigma =_\delta \Omega$ then by (1) there is Ψ such that $\Sigma \rightarrow_\delta \Psi$ and $\Omega \rightarrow_\delta \Psi$. This means that the δ-normal forms of Σ and Ω are equal, so by (2), $\mathrm{nf_d}(\Sigma) \equiv \mathrm{nf_d}(\Omega)$. On the other hand, if $\mathrm{nf_d}(\Sigma) \equiv \mathrm{nf_d}(\Omega)$, then Σ and Ω have the same δ-normal forms (by (2)), so $\Sigma =_\delta \Omega$.
4. By 2, $\rightarrow_\delta$ is weakly normalising. Moreover, Definition 17 of $\mathrm{nf_d}(\Sigma)$ induces an innermost reduction strategy. By a theorem of O'Donnell ([45], or pp. 75–76 of [39]), $\rightarrow_\delta$ is strongly normalising. ⊠

Definition 21
- A book $\mathfrak{B}$ is part of a book $\mathfrak{B}'$, notation $\mathfrak{B} \subseteq \mathfrak{B}'$, if all lines of $\mathfrak{B}$ are lines of $\mathfrak{B}'$.
- A context Γ is part of a context Γ', notation $\Gamma \subseteq \Gamma'$, if all declarations $x{:}\alpha$ of Γ are declarations in Γ'.

Lemma 22 (Weakening) *If* $\mathfrak{B};\Gamma \vdash \Sigma : \Omega$, $\mathfrak{B} \subseteq \mathfrak{B}'$, $\Gamma \subseteq \Gamma'$ *and* $\mathfrak{B}';\Gamma' \vdash \mathrm{OK}$ *then* $\mathfrak{B}';\Gamma' \vdash \Sigma : \Omega$.

PROOF: By induction on the derivation of $\mathfrak{B};\Gamma \vdash \Sigma : \Omega$. ⊠

3 From AUT-68 towards a PTS $\lambda 68$

We want to give a description of AUT-68 within the framework of the Pure Type Systems. One of the most important choices to be made is whether or not to maintain the parameter mechanism (that is: to allow expressions with parameters, as in the second clause of Definition 8). On the one hand, the parameter mechanism is an important feature of AUTOMATH. On the other hand PTSs do not have a parameter mechanism, and the parameter mechanism can be easily imitated by function application (cf. the second clause of the forthcoming Definition 23). Moreover, the description by van Benthem Jutting in [2] of the systems AUT-68 and AUT-QE in a PTS style does not use parameters.

In this paper, we provide a translation to PTSs without parameters. In doing so, we can explain van Benthem Jutting's description of AUT-68 and AUT-QE.

We will see, however, that the way in which we must handle parameters in the resulting PTS is a bit artificial. Moreover, we think that parameters play an important role in the AUTOMATH systems, and that they could play a similar role in other PTSs. Therefore, we present extensions of PTSs with parameters in [32, 41, 33]. These extensions are based on the way in which parameters are handled in AUTOMATH, and it was shown that AUTOMATH can be described very well within these PTSs with parameters.

To describe AUT-68 as a PTS without parameters (call it $\lambda 68$), we first translate the expressions of AUT-68 to typed λ-terms (note that the parameter mechanism of Definition 8 is replaced by repeated function application in PTSs):

Definition 23 Recall that $\mathbb{T}$ and $\mathbb{V}$ are the set of terms and variables for PTSs. We define a mapping $\overline{[\ldots]}$ from the correct expressions in $\mathcal{E}$ (relative to a book $\mathfrak{B}$ and a context Γ) to $\mathbb{T}$. We assume that $\mathcal{C} \cup \mathcal{V} \subseteq \mathbb{V}$.

- $\overline{x} \stackrel{\text{def}}{=} x$ for $x \in \mathcal{V}$;
- $\overline{b(\Sigma_1, \ldots, \Sigma_n)} \stackrel{\text{def}}{=} b\overline{\Sigma_1} \cdots \overline{\Sigma_n}$;
- $\overline{\langle\Omega\rangle\Sigma} \stackrel{\text{def}}{=} \overline{\Sigma}\,\overline{\Omega}$;
- $\overline{\texttt{type}} \stackrel{\text{def}}{=} *$;
- $\overline{[x{:}\Sigma]\Omega} \stackrel{\text{def}}{=} \begin{cases} \Pi x{:}\overline{\Sigma}.\overline{\Omega} & \text{if } [x{:}\Sigma]\Omega \text{ has type } \texttt{type}, \\ \lambda x{:}\overline{\Sigma}.\overline{\Omega} & \text{otherwise} \end{cases}$

With this translation in mind, we want to find a type system $\lambda 68$ that "suits" AUT-68, i.e. if Σ is a correct expression of type Ω with respect to a book $\mathfrak{B}$ and a context Γ, then we want $\mathfrak{B}', \Gamma' \vdash \overline{\Sigma} : \overline{\Omega}$ to be derivable in $\lambda 68$, and vice versa. Here, $\mathfrak{B}'$ and Γ' are some suitable translations of $\mathfrak{B}$ and Γ. The search for a suitable $\lambda 68$ will focus on three points: Π-formation and parameter types; constants and variables; and definitions.

3.1 The choice of the correct formation (Π) rules and the parameter types $\P x{:}A.B$

As $\overline{\texttt{type}} \equiv *$, Definition 15 clarifies which Π-rules are implied by the abstraction mechanism of AUT-68, the rule on the left translates into the rule on the right which is Π-rule $(*, *, *)$ ($\overline{\mathfrak{B}}$ and $\overline{\Gamma}$ are suitable translations of $\mathfrak{B}$ and Γ):

$$\frac{\mathfrak{B};\Gamma \vdash \Sigma_1{:}\texttt{type} \quad \mathfrak{B};\Gamma, x{:}\Sigma_1 \vdash \Omega_1{:}\texttt{type}}{\mathfrak{B};\Gamma \vdash [x{:}\Sigma_1]\Omega_1 : \texttt{type}} \qquad \frac{\overline{\mathfrak{B}},\overline{\Gamma} \vdash \overline{\Sigma_1}: * \quad \overline{\mathfrak{B}},\overline{\Gamma}, x{:}\overline{\Sigma_1} \vdash \overline{\Omega_1}{:}*}{\overline{\mathfrak{B}},\overline{\Gamma} \vdash (\Pi x{:}\overline{\Sigma_1}.\overline{\Omega_1}) : *},$$

It is, however, not immediately clear which Π-rules are induced by the parameter mechanism of AUT-68. Let $\Sigma \equiv b(\Sigma_1, \ldots, \Sigma_n)$ be a correct expression of type Ω with respect to a book $\mathfrak{B}$ and a context Γ. By Definition 14 there is a line $(x_1{:}\alpha_1, \ldots, x_n{:}\alpha_n; b; \Xi_1; \Xi_2)$ in $\mathfrak{B}$ such that each Σ_i is a correct expression with respect to $\mathfrak{B}$ and Γ, and has a type that is definitionally equal to $\alpha_i[x_1, \ldots, x_{i-1}{:=}\Sigma_1, \ldots, \Sigma_{i-1}]$. We also know that $\Omega =_{\beta\mathrm{d}} \Xi_2[x_1, \ldots, x_n{:=}\Sigma_1, \ldots \Sigma_n]$. Now $\overline{\Sigma} \equiv b\overline{\Sigma_1} \cdots \overline{\Sigma_n}$, and, assuming that we can derive in $\lambda 68$ that $\overline{\Sigma_i}$ has type $\overline{\alpha_i}[x_1, \ldots, x_{i-1}{:=}\overline{\Sigma_1}, \ldots, \overline{\Sigma_{i-1}}]$, it is not unreasonable to assign the type $\Pi x_1{:}\overline{\alpha_1} \cdots \Pi x_n{:}\overline{\alpha_n} to b.\overline{\Xi_2}$. We will abbreviate this last term by $\prod_{i=1}^{n} x_i{:}\overline{\alpha_i}.\overline{\Xi_2}$. Then we can derive (using n times the application rule that we will introduce for $\lambda 68$) that $\overline{\Sigma}$ has type $\overline{\Omega}$ in $\lambda 68$.

It is important to notice that the type of b, $\prod_{i=1}^{n} x_i{:}\overline{\alpha_i}.\overline{\Xi_2}$, does not necessarily have an equivalent in AUT-68, as in AUT-68 abstractions over `type` are not allowed (only abstractions over expressions Σ that have `type` as type are possible — cf. Definition 15). In other words, the type of b, $\prod_{i=1}^{n} x_i{:}\overline{\alpha_i}.\overline{\Xi_2}$, is not necessarily a first-class citizen of AUT-68 and should therefore have special treatment in $\lambda 68$. This is the reason to create a special sort $\triangle$, in which these types of AUT-68 constants and definitions are stored. This idea originates from van Benthem Jutting and was firstly presented in [2].

If we construct $\Pi x_n{:}\overline{\alpha_n}.\overline{\Xi_2}$ from $\overline{\Xi_2}$, we must use a rule (s_1, s_2, s_3), where s_1, s_2, s_3 are sorts. Sort s_1 must be the type of $\overline{\alpha_n}$. As $\alpha_n \equiv \mathtt{type}$ or α_n has type $\mathtt{type}$, we must allow the possibilities $s_1 \equiv *$ and $s_1 \equiv \Box$. Similarly, $\Xi_2 \equiv \mathtt{type}$ or Ξ_2 has type $\mathtt{type}$, so we also allow $s_2 \equiv *$ and $s_2 \equiv \Box$. As we intended to store the new type in sort $\triangle$, we take $s_3 \equiv \triangle$.
For similar reasons, we introduce rules $(*, \triangle, \triangle)$ and $(\Box, \triangle, \triangle)$ to construct $\prod_{i=1}^{n} x_i{:}\overline{\alpha_i}.\overline{\Xi_2}$ from $\Pi x_n{:}\overline{\alpha_n}.\overline{\Xi_2}$ for $n > 1$. Hence, we have the Π-rules:
$(*, *, *)$; $(*, *, \triangle)$; $(\Box, *, \triangle)$; $(*, \Box, \triangle)$; $(\Box, \Box, \triangle)$; $(*, \triangle, \triangle)$; $(\Box, \triangle, \triangle)$.

We do not have rules of the form $(\triangle, s_2, s_3)$ or $(s_1, \triangle, s_3)$ with $s_3 \equiv *$ or $s_3 \equiv \Box$. So types of sort $\triangle$ cannot be used to construct types of other sorts. In this way, we can keep the types of the λ-calculus part of AUT-68 separated from the types of the parameter mechanism: the last ones are stored in $\triangle$.

In Example 5.2.4.8 of [2], there is no rule $(*, *, \triangle)$. In principle, this rule is superfluous, as each application of rule $(*, *, \triangle)$ can be replaced by an application of rule $(*, *, *)$. Nevertheless we maintain this rule because:

- The presence of both $(*, *, *)$ and $(*, *, \triangle)$ in the system stresses the fact that AUT-68 has two type mechanisms: one provided by the parameter mechanism and one by the λ-abstraction mechanism;

- There are technical arguments to make a distinction between types formed by the abstraction mechanism and types that appear via the parameter mechanism. In this paper, we denote product types constructed by the abstraction mechanism in the usual way (so: $\Pi x{:}A.B$), whilst we will use the notation $\P x{:}A.B$ for a type constructed by the parameter mechanism. Hence, we have for the constant b above that $b : \P_{i=1}^{n} x_i{:}\overline{\alpha_i}.\overline{\Xi_2}$[10]. As an additional advantage, the resulting system will maintain Unicity of Types. This would have been lost if we use rules $(*, *, *)$ and $(*, *, \triangle)$ without making this difference, as we can then derive both
$$\frac{\alpha{:}* \vdash \alpha{:}* \qquad \alpha{:}*, x{:}\alpha \vdash \alpha{:}*}{\alpha{:}* \vdash (\Pi x{:}\alpha.\alpha) : *} \quad \text{and} \quad \frac{\alpha{:}* \vdash \alpha{:}* \qquad \alpha{:}*, x{:}\alpha \vdash \alpha{:}*}{\alpha{:}* \vdash (\Pi x{:}\alpha.\alpha) : \triangle}$$

- There is another reason to make a distinction between types formed by the abstraction mechanism and types that appear in the translation via the definition mechanism. So far, we use AUT-68 *without* Π-application. In AUT-68 with Π-application (call this system AUT-68Π for the moment; see also Section 5) the application rule of Definition 15 (see below on the left, is replaced by the rule on the right, but the rule describing the type of $b(\Sigma_1, \ldots, \Sigma_n)$ is the same as the rule in Definition 15 (parameters):
$$\frac{\mathfrak{B}; \Gamma \vdash \Sigma_1{:}[x{:}\Omega_1]\Omega_2 \quad \mathfrak{B}; \Gamma \vdash \Sigma_2{:}\Omega_1}{\mathfrak{B}; \Gamma \vdash \langle\Sigma_2\rangle\Sigma_1 : \Omega_2[x{:=}\Sigma_2]} \qquad \frac{\mathfrak{B}; \Gamma \vdash \Sigma_1{:}[x{:}\Omega_1]\Omega_2 \quad \mathfrak{B}; \Gamma \vdash \Sigma_2{:}\Omega_1}{\mathfrak{B}; \Gamma \vdash \langle\Sigma_2\rangle\Sigma_1 : \langle\Sigma_2\rangle\Omega_2}.$$
So if we want to make a translation of AUT-68Π, the application rule for Π-terms has to be different from the application rule for $\P$-terms. Without distinction between Π-terms and $\P$-terms, it would be impossible to amend the system to represent AUT-68Π. Distinguishing between Π-terms and

[10] we use $\P_{i=1}^{n} x_i{:}\overline{\alpha_i}.\overline{\Xi_2}$ as an abbreviation for $\P x_1{:}\overline{\alpha_1} \cdots \P x_n{:}\overline{\alpha_n}.\overline{\Xi_2}$

¶-terms makes it possible to obtain a translation of AUT-68Π from the translation of AUT-68 in a simple way.

3.2 The different treatment of constants and variables

When we seek to translate the AUT-68 judgement $\mathfrak{B};\Gamma\vdash\Sigma:\Omega$ in λ68, we must pay attention to the translation of $\mathfrak{B}$, as there is no equivalent of books in PTSs. Our solution is to store the information on identifiers of $\mathfrak{B}$ in a PTS-context. Therefore, contexts of λ68 will have the form $\Delta;\Gamma$. The left part Δ contains type information on primitive notions and definitions, and can be seen as the translation of the information on primitive notions and definitions in $\mathfrak{B}$. The right part Γ has the usual type information on variables.

The idea to store the constant information of $\mathfrak{B}$ in the left part of the context arises naturally. Let $\mathfrak{B}$ be a correct AUT-68 book, to which we add a line $(\Gamma;b;\text{PN};\Xi_2)$. Then $\Gamma\equiv x_1{:}\alpha_1,\ldots,x_n{:}\alpha_n$ is a correct context with respect to $\mathfrak{B}$, and $\mathfrak{B};\Gamma\vdash\Xi_2{:}\mathtt{type}$ or $\Xi_2\equiv\mathtt{type}$. In λ68 we can work as follows. Assume the information on constants in $\mathfrak{B}$ has been translated into the left part Δ of a λ68 context. We have (assuming that λ68 is a type system that behaves like AUT-68, and writing $\overline{\Gamma}$ for the translation $x_1{:}\overline{\alpha_1},\ldots,x_n{:}\overline{\alpha_n}$ of Γ): $\Delta;\overline{\Gamma}\vdash\overline{\Xi_2}{:}s$ ($s\equiv *$ if $\mathfrak{B};\Gamma\vdash\Xi_2{:}\mathtt{type}$; $s\equiv\Box$ if $\Xi_2\equiv\mathtt{type}$). Applying the ¶-formation rule n times, we obtain $\Delta;\varnothing\vdash\P\overline{\Gamma}.\overline{\Xi_2}:\triangle$ (if Γ is the empty context, then $\P\overline{\Gamma}.\overline{\Xi_2}\equiv\overline{\Xi_2}$, and $\overline{\Xi_2}$ has type $*$ or $\Box$ instead of $\triangle$. We write $\P\overline{\Gamma}$ for $\P_{i=1}^n x_i{:}\overline{\alpha_i}$). As $\P\overline{\Gamma}.\overline{\Xi_2}$ is exactly the type that we want to give to b (see the discussion in Section 3.1), we use this statement as premise for the start rule that introduces b. As the right part $\overline{\Gamma}$ of the original context has disappeared when we applied the ¶-formation rules, $b{:}\P\overline{\Gamma}.\overline{\Xi_2}$ is automatically placed at the righthand end of Δ: The conclusion of the start rule is $\Delta,b{:}\P\overline{\Gamma}.\overline{\Xi_2}\vdash b{:}\P\overline{\Gamma}.\overline{\Xi_2}$.

Adding $b{:}\P\overline{\Gamma}.\overline{\Xi_2}$ at the end of Δ can be compared with adding the line $(\Gamma;b;\text{PN};\Xi_2)$ at the end of $\mathfrak{B}$. This process can be captured by the rule below where $s_1\in\{*,\Box\}$ (compare: $\Xi_2{:}\mathtt{type}$ or $\Xi_2\equiv\mathtt{type}$) and $s_2\in\{*,\Box,\triangle\}$ (usually, $s_2\equiv\triangle$; the cases $s_2\equiv *,\Box$ only occur if Γ is empty):

$$\frac{\Delta;\overline{\Gamma}\vdash\overline{\Xi_2}{:}s_1\qquad\Delta;\vdash\P\overline{\Gamma}.\overline{\Xi_2}{:}s_2}{\Delta,b{:}\P\overline{\Gamma}.\overline{\Xi_2};\vdash b{:}\P\overline{\Gamma}.\overline{\Xi_2}}.$$

3.3 The definition system and the translation using §

A line $(x_1{:}\alpha_1,\ldots,x_n{:}\alpha_n;b;\Xi_1;\Xi_2)$, in which b is a constant and $\Xi_1\in\mathcal{E}$, represents the definition: "for all expressions $\Omega_1,\ldots,\Omega_n$ (obeying some type conditions), $b(\Omega_1,\ldots,\Omega_n)$ abbreviates $\Xi_1[x_1,\ldots,x_n{:=}\Omega_1,\ldots,\Omega_n]$, and has type $\Xi_2[x_1,\ldots,x_n{:=}\Omega_1,\ldots,\Omega_n]$." So in λ68, the context should also mention that $bX_1\cdots X_n$ "is equal to" $\Xi_1[x_1,\ldots,x_n{:=}X_1,\ldots,X_n]$, for all terms $X_1,\ldots,X_n$. This can be done by writing $b{:=}\left(\lambda_{i=1}^n x_i{:}\overline{\alpha_i}.\overline{\Xi_1}\right):\left(\P_{i=1}^n x_i{:}\overline{\alpha_i}.\overline{\Xi_2}\right)$ in the context instead of only $b{:}\P_{i=1}^n x_i{:}\overline{\alpha_i}.\overline{\Xi_2}$, and adding a δ-reduction rule that unfolds the definition of b: if $b{:=}\left(\lambda_{i=1}^n x_i{:}\overline{\alpha_i}.\overline{\Xi_1}\right):\left(\P_{i=1}^n x_i{:}\overline{\alpha_i}.\overline{\Xi_2}\right)\in\Delta$ then $\Delta\vdash b\to_\delta\lambda_{i=1}^n x_i{:}\overline{\alpha_i}.\overline{\Xi_1}$. Unfolding the definition of b in a term $b\overline{\Sigma_1}\cdots\overline{\Sigma_n}$ and applying β-reduction n times gives $\overline{\Xi_1}[x_1{:=}\overline{\Sigma_1}]\cdots[x_n{:=}\overline{\Sigma_n}]$. This procedure corresponds

exactly to the δ-reduction $\Delta \vdash b(\Sigma_1,\ldots,\Sigma_n) \to_\delta \Xi_1[x_1,\ldots,x_n{:=}\Sigma_1,\ldots,\Sigma_n]$ in AUT-68[11]. This method, however, has disadvantages:

- In the AUT-68 line $(x_1{:}\alpha_1,\ldots,x_n{:}\alpha_n; b; \Xi_1; \Xi_2)$, $b(\Sigma_1,\ldots,\Sigma_n)$ has for equivalent in λ68, $b\overline{\Sigma_1}\cdots\overline{\Sigma_n}$. If $n>0$, this λ68-term has $B \equiv b\overline{\Sigma_1}\cdots\overline{\Sigma_m}$ as a subterm for any $m<n$. But B has no equivalent in AUT-68: only after B is applied to suitable terms $\overline{\Sigma_{m+1}},\ldots,\overline{\Sigma_n}$ the result $B\overline{\Sigma_{m+1}}\cdots\overline{\Sigma_n}$ has $b(\Sigma_1,\ldots,\Sigma_n)$ as its equivalent in AUT-68. Hence B is not directly translatable into AUTOMATH, but only an intermediate result necessary to construct the equivalent of $b(\Sigma_1,\ldots,\Sigma_n)$. B is recognisable as an intermediate result via its type $\P^n_{i=m+1}\, x_i{:}\overline{\alpha_i}.\overline{\Xi_2}$, of sort $\triangle$ (not $*$ or $\Box$).

  The method above allows to unfold the definition of b already in B, because $b\overline{\Sigma_1}\cdots\overline{\Sigma_m}$ can reduce to $\left(\lambda^n_{i=1}\, x_i{:}\overline{\alpha_i}.\overline{\Xi_1}\right)\,\overline{\Sigma_1}\cdots\overline{\Sigma_m}$, and we can β-reduce this term m times to $\left(\lambda^n_{i=m+1}\, x_i{:}\overline{\alpha_i}.\overline{\Xi_1}\right)\,[x_j{:=}\overline{\Sigma_j}]^m_{j=1}$. It is more in line with AUT-68 to make such unfolding not possible before *all* n arguments $\overline{\Sigma_1},\ldots,\overline{\Sigma_n}$ have been applied to b, so only when the construction of the equivalent of $b(\Sigma_1,\ldots,\Sigma_n)$ has been completed;

- Moreover, $\lambda^n_{i=1}\, x_i{:}\overline{\alpha_i}.\overline{\Xi_1}$ does not necessarily have an equivalent in AUT-68. Take for example the constant b in line $(\alpha{:}\mathtt{type}; b; [x{:}\alpha]x; [x{:}\alpha]\alpha)$. Then $\lambda^n_{i=1}\, x_i{:}\overline{\alpha_i}.\overline{\Xi_1} \equiv \lambda\alpha{:}*.\lambda x{:}\alpha.x$. Its equivalent in AUT-68 would be $[\alpha{:}\mathtt{type}][x{:}\alpha]x$, but an abstraction $[\alpha{:}\mathtt{type}]$ cannot be made in AUT-68.[12] This explains why we do not incorporate $\lambda^n_{i=1}\, x_i{:}\overline{\alpha_i}.\overline{\Xi_1}$ as a citizen of λ68.

Therefore we choose a different translation. The line $(x_1{:}\alpha_1,\ldots,x_n{:}\alpha_n; b; \Xi_1; \Xi_2)$, where $\Xi_1 \in \mathcal{E}$, will be translated using $b{:=}\left(\S^n_{i=1}\, x_i{:}\overline{\alpha_i}.\overline{\Xi_1}\right) : \left(\P^n_{i=1}\, x_i{:}\overline{\alpha_i}.\overline{\Xi_2}\right)$ instead of $b{:=}\left(\lambda^n_{i=1}\, x_i{:}\overline{\alpha_i}.\overline{\Xi_1}\right) : \left(\P^n_{i=1}\, x_i{:}\overline{\alpha_i}.\overline{\Xi_2}\right)$ in the left part of the translated context Δ. A reduction rule $bX_1\cdots X_n \to_\delta \overline{\Xi_1}[x_1,\ldots,x_n{:=}X_1,\ldots,X_n]$ is added for all terms $X_1,\ldots,X_n$. The symbol $\S$ is used instead of λ. This emphasises that, though both $\S x{:}A$ and $\lambda x{:}A$ are abstractions, they are not the same kind of abstraction.

4 λ68

Here, we give λ68, show that it has the desirable properties of PTSs and that it is the PTS version of AUT-68.

Definition 24 (λ68)

1. Let $\boldsymbol{S}$ is the set of sorts $\{*,\Box,\triangle\}$. Terms of λ68 are given by $\mathcal{T} ::= \mathcal{V} \mid \mathcal{C} \mid \boldsymbol{S} \mid \mathcal{T}\mathcal{T} \mid \lambda\mathcal{V}{:}\mathcal{T}.\mathcal{T} \mid \S\mathcal{V}{:}\mathcal{T}.\mathcal{T} \mid \Pi\mathcal{V}{:}\mathcal{T}.\mathcal{T} \mid \P\mathcal{V}{:}\mathcal{T}.\mathcal{T}$. Free variables $\mathrm{FV}(T)$ and "free" constants $\mathrm{FC}(T)$ of term T are defined as usual;

[11] We can assume that the x_i do not occur in the Σ_j, so the simultaneous substitution $\Xi_1[x_1,\ldots,x_n{:=}\Sigma_1,\ldots,\Sigma_n]$ is equal to $\Xi_1[x_1{:=}\Sigma_1]\cdots[x_n{:=}\Sigma_n]$.

[12] This situation compares to that of Section 3.1, where we found that the type of b is not necessarily a first-class citizen of AUT-68. There, we could not avoid that the type of b became a citizen of λ68 (though we made it a second-class citizen by storing it in the sort $\triangle$).

2. We define the notion of context inductively:

   - $\varnothing;\varnothing$ is a context; $\textsc{dom}\,(\varnothing;\varnothing) = \varnothing$;
   - If $\Delta;\Gamma$ is a context, $x \in \mathcal{V}$, x does not occur in $\Delta;\Gamma$ and $A \in \mathcal{T}$, then $\Delta;\Gamma, x{:}A$ is a context (x is a newly introduced variable); $\textsc{dom}\,(\Delta;\Gamma, x{:}A) = \textsc{dom}\,(\Delta;\Gamma) \cup \{x\}$;
   - If $\Delta;\Gamma$ is a context, $b \in \mathcal{C}$, b does not occur in $\Delta;\Gamma$ and $A \in \mathcal{T}$ then $\Delta, b{:}A;\Gamma$ is a context (in this case b is a *primitive* constant; $\textsc{dom}\,(\Delta, b{:}A;\Gamma) = \textsc{dom}\,(\Delta;\Gamma) \cup \{b\}$;
   - If $\Delta;\Gamma$ is a context, $b \in \mathcal{C}$, b does not occur in $\Delta;\Gamma$, $A \in \mathcal{T}$, and $T \in \mathcal{T}$, then $\Delta, b{:=}T{:}A;\Gamma$ is a context (in this case b is a *defined* constant; $\textsc{dom}\,(\Delta, b{:=}T{:}A;\Gamma) = \textsc{dom}\,(\Delta;\Gamma) \cup \{b\}$.

   Note that a semicolon is used as the separation mark between the two parts of the context. A comma separates expressions within each part.

   We define $\textsc{primcons}\,(\Delta;\Gamma) = \{b \in \textsc{dom}\,(\Delta;\Gamma) \mid b \text{ is a primitive constant}\}$; $\textsc{defcons}\,(\Delta;\Gamma) = \{b \in \textsc{dom}\,(\Delta;\Gamma) \mid b \text{ is a defined constant}\}$; and $\textsc{fv}(\Delta;\Gamma) = \textsc{dom}\,(;\Gamma)$.

3. We define δ-reduction on terms. Let Δ be the left part of a context. If $(b{:=}(\S_{i=1}^{n}\, x_i{:}A_i.T) : (\P_{i=1}^{n}\, x_i{:}A_i.B)) \in \Delta$ and B is not $\P y{:}B_1.B_2$, then $\Delta \vdash bX_1 \cdots X_n \to_\delta T[x_1, \ldots, x_n{:=}X_1, \ldots, X_n]$ for all $X_1, \ldots X_n \in \mathcal{T}$.

   We also have the usual compatibility rules on δ-reduction. We use notations like $\twoheadrightarrow_\delta$, $\twoheadrightarrow_\delta^+$, $=_\delta$ as usual. If no confusion about which Δ occurs, we simply write $bX_1 \cdots X_n \to_\delta T[x_1, \ldots, x_n{:=}X_1, \ldots, X_n]$;

4. We use the usual notion of β-reduction;

5. Judgements in $\lambda 68$ have the form $\Delta;\Gamma \vdash A : B$, where $\Delta;\Gamma$ is a context and A and B are terms. In the case that a judgement $\Delta;\Gamma \vdash A : B$ is derivable according to the rules below, $\Delta;\Gamma$ is a *legal* context and A and B are *legal* terms. We write $\Delta;\Gamma \vdash A : B : C$ if both $\Delta;\Gamma \vdash A : B$ and $\Delta;\Gamma \vdash B : C$ are derivable in $\lambda 68$. The rules for $\lambda 68$ are given in Figure 5 (v, pc, and dc are shorthand for variable, primitive constant, and defined constant, respectively). The newly introduced variables in the Start-rules and Weakening-rules are assumed to be fresh. Moreover, when introducing a variable x with a "pc"-rule or a "dc"-rule, we assume $x \in \mathcal{C}$, and when introducing x via a "v"-rule, we assume $x \in \mathcal{V}$. We write $\Delta;\Gamma \vdash_{\lambda 68} A : B$ instead of $\Delta;\Gamma \vdash A : B$ if the latter gives rise to confusion.

Note that there is no rule (§). This is because we do not want terms like $\S\, x{:}A.B$ to be first-class citizens of $\lambda 68$: they do not have an equivalent in AUTOMATH.

Definition 25 We define: $\Delta_1;\Gamma_1 \vdash \Delta_2;\Gamma_2$ if and only if
- If $b{:}A \in \Delta_2;\Gamma_2$ then $\Delta_1;\Gamma_1 \vdash b{:}A$;
- If $b{:=}T{:}A \in \Delta_2$ then $\Delta_1;\Gamma_1 \vdash b{:}A$;
- If $b{:=}(\S_{i=1}^{n}\, x_i : A_i.U){:}B \in \Delta_2$ and $U \not\equiv \S\, y{:}B.A'$ then $\Delta_1 \vdash bx_1 \cdots x_n =_{\beta\delta} U$.

$$(\textbf{Axiom}) \qquad ;\vdash * : \Box$$

$$(\textbf{Start} : \textbf{v}) \qquad \frac{\Delta;\Gamma \vdash A : s}{\Delta;\Gamma, x{:}A \vdash x : A} \qquad s \equiv *, \Box$$

$$(\textbf{Start} : \textbf{pc}) \qquad \frac{\Delta;\Gamma \vdash B : s_1 \qquad \Delta;\vdash \P\,\Gamma.B : s_2}{\Delta, b{:}\P\,\Gamma.B;\vdash b : \P\,\Gamma.B} \qquad s_1 \equiv *, \Box$$

$$(\textbf{Start} : \textbf{dc}) \qquad \frac{\Delta;\Gamma \vdash T : B : s_1 \qquad \Delta;\vdash \P\,\Gamma.B : s_2}{\Delta, b{:=}(\S\,\Gamma.T){:}(\P\,\Gamma.B);\vdash b : \P\,\Gamma.B} \qquad s_1 \equiv *, \Box$$

$$(\textbf{Weak} : \textbf{v}) \qquad \frac{\Delta;\Gamma \vdash M : N \qquad \Delta;\Gamma \vdash A : s}{\Delta;\Gamma, x{:}A \vdash M : N} \qquad s \equiv *, \Box$$

$$(\textbf{Weak} : \textbf{pc}) \qquad \frac{\Delta;\vdash M : N \;\; \Delta;\Gamma \vdash B : s_1 \;\; \Delta;\vdash \P\,\Gamma.B : s_2}{\Delta, b{:}\P\,\Gamma.B;\vdash M : N} \qquad s_1 \equiv *, \Box$$

$$(\textbf{Weak} : \textbf{dc}) \qquad \frac{\Delta;\vdash M : N \;\; \Delta;\Gamma \vdash T : B : s_1 \;\; \Delta;\vdash \P\,\Gamma.B : s_2}{\Delta, b{:=}(\S\,\Gamma.T){:}(\P\,\Gamma.B);\vdash M : N} \qquad s_1 \equiv *, \Box$$

$$(\mathbf{\Pi - form}) \qquad \frac{\Delta;\Gamma \vdash A : * \qquad \Delta;\Gamma, x{:}A \vdash B : *}{\Delta;\Gamma \vdash (\Pi x{:}A.B) : *}$$

$$(\mathbf{\P - form}) \qquad \frac{\Delta;\Gamma \vdash A : s_1 \qquad \Delta;\Gamma, x{:}A \vdash B : s_2}{\Delta;\Gamma \vdash (\P x{:}A.B) : \triangle} \qquad s_1 \equiv *, \Box$$

$$(\boldsymbol{\lambda}) \qquad \frac{\Delta;\Gamma \vdash \Pi x{:}A.B : * \qquad \Delta;\Gamma, x{:}A \vdash F : B}{\Delta;\Gamma \vdash (\lambda x{:}A.F) : (\Pi x{:}A.B)}$$

$$(\textbf{App}_1) \qquad \frac{\Delta;\Gamma \vdash M : \Pi x{:}A.B \qquad \Delta;\Gamma \vdash N : A}{\Delta;\Gamma \vdash MN : B[x{:=}N]}$$

$$(\textbf{App}_2) \qquad \frac{\Delta;\Gamma \vdash M : \P x{:}A.B \qquad \Delta;\Gamma \vdash N : A}{\Delta;\Gamma \vdash MN : B[x{:=}N]}$$

$$(\textbf{Conv}) \qquad \frac{\Delta;\Gamma \vdash M : A \qquad \Delta;\Gamma \vdash B : s \qquad \Delta \vdash A =_{\beta\delta} B}{\Delta;\Gamma \vdash M : B}$$

Figure 4: Rules of $\lambda 68$

Many properties for PTSs hold for $\lambda 68$ and can be proved by the same methods as for PTSs. Due to the split of contexts and the different treatment of constants and variables, these properties are on some points differently formulated than usual. The proofs of Lemmas 26, 27, 30, 31, 32 follow [2].

Lemma 26 (Free Variable Lemma) *Let $\Delta \equiv b_1{:}B_1, \ldots, b_m{:}B_m$ (in Δ, also expressions $b_i{:=}T_i{:}B_i$ may occur, but for uniformity of notation we leave out the $:=T_i$-part); let $\Gamma \equiv x_1{:}A_1, \ldots, x_n{:}A_n$ and $\Delta;\Gamma \vdash M : N$. Then:*

- *The $b_1, \ldots, b_m \in \mathcal{C}$ and $x_1, \ldots, x_n \in \mathcal{V}$ are all distinct;*
- *$\mathrm{FC}(M), \mathrm{FC}(N) \subseteq \{b_1, \ldots, b_m\}$; $\mathrm{FV}(M), \mathrm{FV}(N) \subseteq \{x_1, \ldots, x_n\}$;*
- *$b_1{:}B_1, \ldots, b_{i-1}{:}B_{i-1}; \vdash B_i{:}s_i$ for $s_i \in \{*, \Box, \triangle\}$; and $\Delta; x_1{:}A_1, \ldots, x_{j-1}{:}A_{j-1} \vdash A_j{:}t_j$ for $t_j \in \{*, \Box\}$.*

Lemma 27

- **(Start)** *Let $\Delta;\Gamma$ be a legal context. Then $\Delta;\Gamma \vdash * : \Box$, and if $b{:}A \in \Delta;\Gamma$, or $c{:=}T{:}A \in \Delta$, then $\Delta;\Gamma \vdash c : A$.*
- **(Definition)** *Assume $\Delta_1, b{:=}(\S_{i=1}^n x_i{:}A_i.T) : (\P_{i=1}^n x_i{:}A_i.B)$ and $\Delta_2;\Gamma \vdash M : N$, where B is not of the form $\P y{:}B_1.B_2$. Then $\Delta_1; x_1{:}A_1, \ldots, x_n{:}A_n \vdash T : B : s$ for an $s \in \{*, \Box\}$.*

The Transitivity Lemma must be formulated differently than usual (cf. 30) because contexts may contain definitions. To the usual formulation

> *"Let $\Delta_1;\Gamma_1$ and $\Delta_2;\Gamma_2$ be contexts, of which $\Delta_1;\Gamma_1$ is legal. Assume that for all $b{:}A \in \Delta_2;\Gamma_2$ and for all $b{:=}T{:}A \in \Delta_2;\Gamma_2$, $\Delta_1;\Gamma_1 \vdash b{:}A$. Then $\Delta_2;\Gamma_2 \vdash B : C \Rightarrow \Delta_1;\Gamma_1 \vdash B : C$."*

we must add a clause that b is defined in $\Delta_1;\Gamma_1$ in a similar way as it has been defined in $\Delta_2;\Gamma_2$. The next example shows that things go wrong otherwise:

Example 28 Let $\Delta_1 \equiv \mathtt{b_1}{:}*, \mathtt{b_2}{:}*, \mathtt{b_3}{:=}\mathtt{b_1}{:}*$ and $\Delta_2 \equiv \mathtt{b_1}{:}*, \mathtt{b_2}{:}*, \mathtt{b_3}{:=}\mathtt{b_2}{:}*$. Let $\Gamma_1 \equiv \Gamma_2 \equiv \mathtt{x_3}{:}\mathtt{b_3}$. Note that all the assumptions of the traditional formulation of the Transitivity Lemma (see above) hold for $\Delta_1;\Gamma_1$ and $\Delta_2;\Gamma_2$. Nevertheless, we can derive $\Delta_2;\Gamma_2 \vdash \mathtt{x_3} : \mathtt{b_2}$ (because $\Delta_2;\Gamma_2 \vdash \mathtt{x}{:}\mathtt{b_3}$ and according to Δ_2, $\mathtt{b_3} =_{\beta\mathrm{d}} \mathtt{b_2}$, so we can use the conversion rule). But we cannot derive $\Delta_1;\Gamma_1 \vdash \mathtt{x_3} : \mathtt{b_2}$ (because $\mathtt{b_3}$ and $\mathtt{b_2}$ are *not* definitionally equal according to Δ_1).

The following formulation of the Transitivity Lemma is correct:

Definition 29 We define: $\Delta_1;\Gamma_1 \vdash \Delta_2;\Gamma_2$ if and only if

- If $b{:}A \in \Delta_2;\Gamma_2$ then $\Delta_1;\Gamma_1 \vdash b{:}A$;
- If $b{:=}T{:}A \in \Delta_2$ then $\Delta_1;\Gamma_1 \vdash b{:}A$;
- If $b{:=}(\S_{i=1}^n x_i : A_i.U){:}B \in \Delta_2$ and $U \not\equiv \S\, y{:}B.A'$ then $\Delta_1 \vdash bx_1 \cdots x_n =_{\beta\delta} U$.

Lemma 30

- **(Transitivity)** *Assume* $\Delta_1;\Gamma_1 \vdash \Delta_2;\Gamma_2$ *and* $\Delta_2;\Gamma_2 \vdash B : C$. *Then* $\Delta_1;\Gamma_1 \vdash B : C$.
- **(Substitution)** *If* $\Delta;\Gamma_1, x{:}A, \Gamma_2 \vdash B : C$ *and* $\Delta;\Gamma_1 \vdash D : A$ *then* $\Delta;\Gamma_1, \Gamma_2[x{:=}D] \vdash B[x{:=}D] : C[x{:=}D]$.
- **(Thinning)** *Let* $\Delta_1;\Gamma_1$ *be a legal context, and let* $\Delta_2;\Gamma_2$ *be a legal context such that* $\Delta_1 \subseteq \Delta_2$ *and* $\Gamma_1 \subseteq \Gamma_2$. *Then* $\Delta_1;\Gamma_1 \vdash A : B \Rightarrow \Delta_2;\Gamma_2 \vdash A : B$.

Lemma 31 (Generation Lemma)

- *If* $x \in \mathcal{V}$ *and* $\Delta;\Gamma \vdash x{:}C$ *then* $\exists s \in \{*, \Box\}$ *and* $B =_{\beta\delta} C$ *such that* $\Delta;\Gamma \vdash B : s$ *and* $x{:}B \in \Gamma$;
- *If* $b \in \mathcal{C}$ *and* $\Delta;\Gamma \vdash b{:}C$ *then* $\exists s \in \boldsymbol{S}$ *and* $B =_{\beta\delta} C$ *such that* $\Delta;\Gamma \vdash B : s$, *and either* $b{:}B \in \Delta$ *or* $\exists T$ *such that* $b{:=}T{:}B \in \Delta$;
- *If* $s \in \boldsymbol{S}$ *and* $\Delta;\Gamma \vdash s{:}C$ *then* $s \equiv *$ *and* $C =_{\beta\delta} \Box$;
- *If* $\Delta;\Gamma \vdash MN : C$ *then* $\exists A, B$ *such that* $\Delta;\Gamma \vdash M : (\Pi x{:}A.B)$ *or* $\Delta;\Gamma \vdash M : (\P x{:}A.B)$, *and* $\Delta;\Gamma \vdash N{:}A$ *and* $C =_{\beta\delta} B[x{:=}N]$;
- *If* $\Delta;\Gamma \vdash (\lambda x{:}A.b) : C$ *then* $\exists B$ *such that* $\Delta;\Gamma \vdash (\Pi x{:}A.B) : *$, $\Delta;\Gamma, x{:}A \vdash b : B$ *and* $C =_{\beta\delta} \Pi x{:}A.B$;
- *Assume* $\Delta;\Gamma \vdash (\Pi x{:}A.B) : C$. *Then* $C =_{\beta\delta} *$, $\Delta;\Gamma \vdash A{:}*$ *and* $\Delta;\Gamma, x{:}A \vdash B{:}*$;
- *If* $\Delta;\Gamma \vdash (\P x{:}A.B) : C$ *then* $C =_{\beta\delta} \triangle$, $\Delta;\Gamma \vdash A{:}s_1$ *for* $s_1 \in \{*, \Box\}$, *and* $\Delta;\Gamma, x{:}A \vdash B{:}s_2$ *for* $s_2 \in \{*, \Box, \triangle\}$.

Lemma 32

- **(Unicity of Types)** *If* $\Delta;\Gamma \vdash A : B_1$ *and* $\Delta;\Gamma \vdash A : B_2$ *then* $B_1 =_{\beta\delta} B_2$.
- **(Correctness of Types)** *If* $\Delta;\Gamma \vdash A : B$ *then there is* $s \in \boldsymbol{S}$ *such that* $B \equiv s$ *or* $\Delta;\Gamma \vdash B : s$.
- *If* $\Delta;\Gamma \vdash A : (\Pi x{:}B_1.B_2)$ *then* $\Delta;\Gamma \vdash B_1 : *$; *and* $\Delta;\Gamma, x{:}B_1 \vdash B_2 : *$.
- *If* $\Delta;\Gamma \vdash A : (\P x{:}B_1.B_2)$ *then* $\Delta;\Gamma \vdash B_1 : s_1$ *for* $s_1 \in \{*, \Box\}$; *and* $\Delta;\Gamma, x{:}B_1 \vdash B_2{:}s_2$ *for some* s_2.

In order to show some properties of the reduction relations $\to_\beta$, $\to_\delta$ and $\to_{\beta\delta}$ and as δ-reduction also depends on books, we first have to give a translation of AUT-68 books and AUT-contexts to λ68-contexts:

Definition 33

- Let Γ be a AUT-68-context $x_1{:}\alpha_1, \ldots, x_n{:}\alpha_n$. Then $\overline{\Gamma} \stackrel{\text{def}}{=} x_1{:}\overline{\alpha_1}, \ldots, x_n{:}\overline{\alpha_n}$.
- Let $\mathfrak{B}$ be a book. We define the left part $\overline{\mathfrak{B}}$ of a context in λ68 as:
  - $\overline{\varnothing} \stackrel{\text{def}}{=} \varnothing$;
  - $\overline{\mathfrak{B}, (\Gamma; x; —; \Omega)} \stackrel{\text{def}}{=} \overline{\mathfrak{B}}$;
  - $\overline{\mathfrak{B}, (\Gamma; b; \text{PN}; \Omega)} \stackrel{\text{def}}{=} \overline{\mathfrak{B}}, b{:}\,\P\,\overline{\Gamma}.\overline{\Omega}$;
  - $\overline{\mathfrak{B}, (\Gamma; b; \Sigma; \Omega)} \stackrel{\text{def}}{=} \overline{\mathfrak{B}}, b{:=}\,\S\,\overline{\Gamma}.\overline{\Sigma}{:}\,\P\,\overline{\Gamma}.\overline{\Omega}$.

```
  prop    :   *,
   and    :   ¶x:prop.¶y:prop.prop,
 proof    :   ¶x:prop.*,
 and-I    :   ¶x:prop.¶y:prop.¶px:(proof)x.¶py:(proof)y.(proof)((and)xy),
and-O1    :   ¶x:prop.¶y:prop.¶pxy:(proof)((and)xy).(proof)x,
and-O2    :   ¶x:prop.¶y:prop.¶pxy:(proof)((and)xy).(proof)y,
 and-R    :=  §x:prop.§prx : (proof)x.(and-I)xx(prx)(prx)
           :   ¶x:prop.¶prx : (proof)x.(proof)((and)xx),
 and-S    :=  §x:prop.§y:prop.§pxy:(proof)((and)xy).
               (and-I)yx((and-O2)xy(pxy))((and-O1)xy(pxy))
           :   ¶x:prop.¶y:prop.¶pxy:(proof)((and)xy).(proof)((and)yx)
```

Figure 5: Translation of Example 13

Example 34 The translation of the AUTOMATH book of Example 13 into $\lambda 68$ is given in Figure 5. (Because of the habit in computer science to use more than one digit for a variable, we have to write some additional brackets around subterms like `proof` to preserve unambiguity). Note that all variable declarations of the original book have disappeared in the translation. In the original book, they do not add any new knowledge but are only used to construct contexts. In our translation, this happens in the right (instead of the left) part of the context.

Lemma 35 *Assume, Σ is a correct expression with respect to a book $\mathfrak{B}$.*
• *1. $\Sigma \to_\beta \Sigma'$ if and only if $\overline{\Sigma} \to_\beta \overline{\Sigma'}$;*
• *2. $\mathfrak{B} \vdash_{AUT-68} \Sigma \to_\delta \Sigma'$ if and only if $\overline{\mathfrak{B}} \vdash_{\lambda 68} \overline{\Sigma} \to_\delta \overline{\Sigma'}$.*

PROOF: An easy induction on the structure of Σ. ⊠

The Church-Rosser property of $\to_{\beta\delta}$ (Theorem 44) will be proved by Parallel Reduction $\Rightarrow_{\beta\delta}$, à la Martin-Löf and Tait (see Section 3.2 of [1]). The next three pages are devoted to this proof. We use IH for Induction Hypothesis.

Definition 36 Let Δ be the left part of a context. We define a "parallel reduction" relation $\Rightarrow_{\beta\delta}$ on $\mathcal{T}$:
• For $x \in \mathcal{V}$, $\Delta \vdash x \Rightarrow_{\beta\delta} x$;
• For $b \in \mathcal{C}$, $\Delta \vdash b \Rightarrow_{\beta\delta} b$;
• For $s \in \boldsymbol{S}$, $\Delta \vdash s \Rightarrow_{\beta\delta} s$;
• If $\Delta \vdash P \Rightarrow_{\beta\delta} P'$ and $\Delta \vdash Q \Rightarrow_{\beta\delta} Q'$, then
  ∘ $\Delta \vdash \lambda x{:}P.Q \Rightarrow_{\beta\delta} \lambda x{:}P'.Q'$;
  ∘ $\Delta \vdash \Pi x{:}P.Q \Rightarrow_{\beta\delta} \Pi x{:}P'.Q'$;
  ∘ $\Delta \vdash \P x{:}P.Q \Rightarrow_{\beta\delta} \P x{:}P'.Q'$;
  ∘ $\Delta \vdash PQ \Rightarrow_{\beta\delta} P'Q'$;
• If $\Delta \vdash Q \Rightarrow_{\beta\delta} Q'$ and $\Delta \vdash R \Rightarrow_{\beta\delta} R'$, then $\Delta \vdash (\lambda x{:}P.Q)R \Rightarrow_{\beta\delta} Q'[x{:=}R']$;
• If $b{:=}(\S_{i=1}^n\, x_i{:}A_i.T){:}(\P_{i=1}^n\, x_i{:}A_i.U) \in \Delta$, the term T is not of the form $\S y{:}T_1.T_2$, $\Delta \vdash T \Rightarrow_{\beta\delta} T'$ and $\Delta \vdash M_i \Rightarrow_{\beta\delta} M_i'$ for $i = 1, \ldots, n$, then $\Delta \vdash bM_1 \cdots M_n \Rightarrow_{\beta\delta} T'[x_1, \ldots, x_n{:=}M_1', \ldots, M_n']$.

Some elementary properties of $\Rightarrow_{\beta\delta}$ are:

Lemma 37 (Properties of $\Rightarrow_{\beta\delta}$) *Let Δ be the left part of a context. Then:*
1. $\Delta \vdash M \Rightarrow_{\beta\delta} M$; *2. If $\Delta \vdash M \to_{\beta\delta} M'$ then $\Delta \vdash M \Rightarrow_{\beta\delta} M'$;*
3. If $\Delta \vdash M \Rightarrow_{\beta\delta} M'$ then $\Delta \vdash M \twoheadrightarrow_{\beta\delta} M'$.

PROOF: All proofs can be given by induction on the structure of M. ⊠

By Lemma 37, $\twoheadrightarrow_{\beta\delta}$ (the reflexive and transitive closure of $\rightarrow_{\beta\delta}$) in Δ is the same relation as the reflexive and transitive closure of $\Rightarrow_{\beta\delta}$ in Δ. Therefore, if we want to prove Church-Rosser $\twoheadrightarrow_{\beta\delta}$, it suffices to prove the Diamond Property for $\Rightarrow_{\beta\delta}$. We first make some preliminary definitions and remarks:

Lemma 38 *If* $\Delta \vdash M \Rightarrow_{\beta\delta} M'$ *and* $\Delta \vdash N \Rightarrow_{\beta\delta} N'$ *then* $\Delta \vdash M[y{:=}N] \Rightarrow_{\beta\delta} M'[y{:=}N']$.

PROOF: Induction on the structure of M. ⊠

Lemma 39 *Assume,* Δ *and* Δ, Δ' *are left parts of legal contexts, and* $\mathrm{FC}(M) \subseteq \mathrm{DOM}(\Delta)$. *Then* $\Delta \vdash M \Rightarrow_{\beta\delta} N$ *if and only if* $\Delta, \Delta' \vdash M \Rightarrow_{\beta\delta} N$.

PROOF: By induction on the length of Δ and by induction on the definition of $\Delta \vdash M \Rightarrow_{\beta\delta} N$. All cases in the definition of $\Delta \vdash M \Rightarrow_{\beta\delta} N$ follow directly from IH for $\Delta \vdash M \Rightarrow_{\beta\delta} N$, except for the case $bM_1 \cdots M_n \Rightarrow_{\beta\delta} T'[x_1, \ldots, x_n{:=}M'_1, \ldots, M'_n]$. As $\mathrm{FC}(M) \subseteq \mathrm{DOM}(\Delta)$, we have $b \in \mathrm{DOM}(\Delta)$.
Write $\Delta \equiv \Delta_1, b{:=}(\S_{i=1}^n x_i{:}A_i.T){:}(\P_{i=1}^n x_i{:}A_i.U), \Delta_2$.

- Notice that T is typable in $\Delta_1; x_1{:}A_1, \ldots, x_n{:}A_n$ (Definition Lemma). By the Free Variable Lemma: $\mathrm{FC}(T) \subseteq \mathrm{DOM}(\Delta_1)$. By IH on the length of Δ we have $\Delta_1 \vdash T \Rightarrow_{\beta\delta} T'$ iff $\Delta \vdash T \Rightarrow_{\beta\delta} T'$, and $\Delta_1 \vdash T \Rightarrow_{\beta\delta} T'$ iff $\Delta, \Delta' \vdash T \Rightarrow_{\beta\delta} T'$;
- We conclude: $\Delta \vdash T \Rightarrow_{\beta\delta} T'$ iff $\Delta, \Delta' \vdash T \Rightarrow_{\beta\delta} T'$;
- By IH on the definition of $\Delta \vdash M \Rightarrow_{\beta\delta} N$, we have $\Delta \vdash M_i \Rightarrow_{\beta\delta} M'_i$ iff $\Delta, \Delta' \vdash M_i \Rightarrow_{\beta\delta} M'_i$;
- Note that $b{:=}(\S_{i=1}^n x_i{:}A_i.T){:}(\P_{i=1}^n x_i{:}A_i.U)$ is an element of both Δ, Δ' and Δ. Moreover, $b \notin \mathrm{DOM}(\Delta')$ (as Δ, Δ' is the left part of a *legal* context). Hence $\Delta \vdash bM_1 \cdots M_n \Rightarrow_{\beta\delta} N$ iff $\Delta, \Delta' \vdash bM_1 \cdots M_n \Rightarrow_{\beta\delta} N$. ⊠

For left parts Δ of contexts and for $M \in \mathcal{T}$ with $\mathrm{FC}(M) \subseteq \mathrm{DOM}(\Delta)$, we define a term M^Δ. In M^Δ, all β-redexes that exist in M are contracted simultaneously (this is a usual step in a proof of Church-Rosser by Parallel Reduction), but also all δ-redexes are contracted. We will show that $\Delta \vdash N \Rightarrow_{\beta\delta} M^\Delta$ for any N with $\Delta \vdash M \Rightarrow_{\beta\delta} N$; so M^Δ helps us to show the Diamond Property for $\Rightarrow_{\beta\delta}$.

Definition 40 We define M^Δ for any left part Δ of a context and any $M \in \mathcal{T}$ such that $\mathrm{FC}(M) \subseteq \mathrm{DOM}(\Delta)$. The definition of M^Δ is by induction on the length of Δ. So assume $M^{\Delta'}$ has been defined for contexts Δ' shorter than Δ. We use induction on the structure of M:

- $x^\Delta \stackrel{\mathrm{def}}{=} x$ for any $x \in \mathcal{V}$; $\quad s^\Delta \stackrel{\mathrm{def}}{=} s$ for any $s \in \boldsymbol{S}$;
- $M \equiv b$. Distinguish:
  - $b^\Delta \stackrel{\mathrm{def}}{=} b$ for any $b \in \mathrm{PRIMCONS}(\Delta;)$;
  - $b^\Delta \stackrel{\mathrm{def}}{=} b$ for any $b \in \mathrm{DEFCONS}(\Delta;)$ that is not a δ-redex;

- If $b \in \textsc{defcons}(\Delta;)$ is a δ-redex, then $\Delta \equiv \Delta_1, b{:=}T{:}U, \Delta_2$, where $T \not\equiv \S y{:}T_1.T_2$. By the Definition Lemma, $\Delta_1; \vdash T : U$, so we can assume that T^{Δ_1} has already been defined. Then $b^\Delta \stackrel{\text{def}}{=} T^{\Delta_1}$;

- $(\lambda x{:}P.Q)^\Delta \stackrel{\text{def}}{=} \lambda x{:}P^\Delta.Q^\Delta$; $\quad (\Pi x{:}P.Q)^\Delta \stackrel{\text{def}}{=} \Pi x{:}P^\Delta.Q^\Delta$;
  $(\P x{:}P.Q)^\Delta \stackrel{\text{def}}{=} \P x{:}P^\Delta.Q^\Delta$;

- M is an application term. We distinguish three possibilities:
  - $M \equiv PQ$ is not a $\beta\delta$-redex. Then we define $M^\Delta \stackrel{\text{def}}{=} P^\Delta Q^\Delta$;
  - M is a β-redex $(\lambda x{:}P.Q)R$. We define $M^\Delta \stackrel{\text{def}}{=} Q^\Delta[x{:=}R^\Delta]$;
  - M is a δ-redex $bM_1 \cdots M_n$, and for T is not of the form $\S y{:}T_1.T_2$, Δ is $\Delta_1, b{:=}\left(\S_{i=1}^n\, x_i{:}A_i.T\right) : \left(\P_{i=1}^n\, x_i{:}A_i.U\right), \Delta_2$. So $\Delta_1; x_1{:}A_1, \ldots, x_n{:}A_n \vdash T : U$ (by the Definition Lemma) and we can assume that T^{Δ_1} has already been defined. Then $M^\Delta \stackrel{\text{def}}{=} T^{\Delta_1}[x_1, \ldots, x_n{:=}M_1^\Delta, \ldots, M_n^\Delta]$.

Lemma 41 *Let Δ be the left part of a legal context. Then $\Delta \vdash M \Rightarrow_{\beta\delta} M^\Delta$ for all M with $\textsc{fc}(M) \subseteq \textsc{dom}(\Delta)$.*

Proof: By induction on the definition of M^Δ. We only treat the case $\Delta \vdash bM_1 \cdots M_n \Rightarrow_{\beta\delta} (bM_1 \cdots M_n)^\Delta$ where $bM_1 \cdots M_n$ is a δ-redex. Write $\Delta \equiv \Delta_1, b{:=}\left(\S_{i=1}^n\, x_i{:}A_i.T\right) : \left(\P_{i=1}^n\, x_i{:}A_i.U\right), \Delta_2$, as in the definition of $(bM_1 \cdots M_n)^\Delta$. By induction, we may assume that $\Delta_1 \vdash T \Rightarrow_{\beta\delta} T^{\Delta_1}$ and $\Delta \vdash M_i \Rightarrow_{\beta\delta} M_i^\Delta$. By the Definition Lemma, T is typable in $\Delta_1; x_1{:}A_1, \ldots, x_n{:}A_n$, so by the Free Variable Lemma, $\textsc{fc}(T) \subseteq \textsc{dom}(\Delta_1)$. By Lemma 39, $\Delta \vdash T \Rightarrow_{\beta\delta} T^{\Delta_1}$. So $\Delta \vdash bM_1 \cdots M_n \Rightarrow_{\beta\delta} T^{\Delta_1}[x_1, \ldots, x_n{:=}M_1^\Delta, \ldots, M_n^\Delta]$. ⊠

Theorem 42 *Let Δ be the left part of a legal context. Assume $\textsc{fc}(M) \subseteq \textsc{dom}(\Delta)$. If $\Delta \vdash M \Rightarrow_{\beta\delta} N$ then $\Delta \vdash N \Rightarrow_{\beta\delta} M^\Delta$.*

Proof: Induction on the the definition of M^Δ.

- $M \equiv x$. Then $N \equiv x$ and $M^\Delta \equiv x$;
- $M \equiv b$. Distinguish:
  - $b \in \textsc{primcons}(\Delta;)$. Then $N \equiv b$ and $M^\Delta \equiv b$;
  - $b \in \textsc{defcons}(\Delta;)$, but b is not a δ-redex. Then $N \equiv b$ and $M^\Delta \equiv b$;
  - $b \in \textsc{defcons}(\Delta;)$, and $\Delta \equiv \Delta_1, b{:=}T{:}U, \Delta_2$, and $T \not\equiv \S y{:}T_1.T_2$. Then either $N \equiv b$ or $N \equiv T'$ where $T \Rightarrow_{\beta\delta} T'$. If $N \equiv b$ then $M \equiv N$ and we can use Lemma 41. If $N \equiv T$ then observe that by IH, $\Delta_1 \vdash T \Rightarrow_{\beta\delta} T^{\Delta_1}$, that by Lemma 39 $\Delta \vdash T \Rightarrow_{\beta\delta} T^{\Delta_1}$, and that $M^\Delta \equiv T^{\Delta_1}$;
- $M \equiv s$. Then $N \equiv s$ and $M^\Delta \equiv s$;
- $M \equiv \lambda x{:}P.Q$. Then $N \equiv \lambda x{:}P'.Q'$ for some P', Q' with $\Delta \vdash P \Rightarrow_{\beta\delta} P'$ and $\Delta \vdash Q \Rightarrow_{\beta\delta} Q'$. By IH on P and Q we find $\Delta \vdash P' \Rightarrow_{\beta\delta} P^\Delta$ and $\Delta \vdash Q' \Rightarrow_{\beta\delta} Q^\Delta$. Therefore $\Delta \vdash \lambda x{:}P'.Q' \Rightarrow_{\beta\delta} \lambda x{:}P^\Delta.Q^\Delta$.
  The cases $M \equiv \Pi x{:}P.Q$, $M \equiv \P x{:}P.Q$, and $M \equiv PQ$ where PQ is not a $\beta\delta$-redex, are proved similarly;

- M is an application term (and is either a β or a δ-redex). Distinguish:
  - M is a β-redex, $M \equiv (\lambda x{:}P.Q)R$. Distinguish:
    - $N \equiv (\lambda x{:}P'.Q')R'$ for P', Q', R' with $\Delta \vdash P \Rightarrow_{\beta\delta} P'$, $\Delta \vdash Q \Rightarrow_{\beta\delta} Q'$ and $\Delta \vdash R \Rightarrow_{\beta\delta} R'$. By induction, $\Delta \vdash Q' \Rightarrow_{\beta\delta} Q^\Delta$ and $\Delta \vdash R' \Rightarrow_{\beta\delta} R^\Delta$. Therefore $\Delta \vdash N \Rightarrow_{\beta\delta} Q^\Delta[x{:=}R^\Delta]$;
    - $N \equiv Q'[x{:=}R']$ for Q', R' with $\Delta \vdash Q \Rightarrow_{\beta\delta} Q'$ and $\Delta \vdash R \Rightarrow_{\beta\delta} R'$. By induction, $\Delta \vdash Q' \Rightarrow_{\beta\delta} Q^\Delta$ and $\Delta \vdash R' \Rightarrow_{\beta\delta} R^\Delta$. By Lemma 38, $\Delta \vdash Q'[x{:=}R'] \Rightarrow_{\beta\delta} Q^\Delta[x{:=}R^\Delta]$;
  - M is a δ-redex, $M \equiv bM_1 \cdots M_n$, and for $T \not\equiv \S y{:}T_1.T_2$, we have $\Delta \equiv \Delta_1, b{:=}(\S_{i=1}^n x_i{:}A_i.T) : (\P_{i=1}^n x_i{:}A_i.U), \Delta_2$.
    - $N \equiv bM_1' \cdots M_n'$ for M_i' with $\Delta \vdash M_i \Rightarrow_{\beta\delta} M_i'$. By induction, we have $\Delta \vdash M_i' \Rightarrow_{\beta\delta} M_i^\Delta$. By the Definition Lemma, T is typable in a context $\Delta_1; x_1{:}A_1, \ldots, x_n{:}A_n$, so by the Free Variable Lemma, $\mathrm{FC}(T) \subseteq \mathrm{DOM}(\Delta_1)$. By Lemma 41, $\Delta_1 \vdash T \Rightarrow_{\beta\delta} T^{\Delta_1}$. By Lemma 39, $\Delta \vdash T \Rightarrow_{\beta\delta} T^{\Delta_1}$. Hence $\Delta \vdash N \Rightarrow_{\beta\delta} T^{\Delta_1}[x_1, \ldots, x_n{:=}M_1^\Delta, \ldots, M_n^\Delta]$;
    - $N \equiv T'[x_1, \ldots, x_n{:=}M_1', \ldots, M_n']$ for a T' with $\Delta \vdash T \Rightarrow_{\beta\delta} T'$ and for M_i' with $\Delta \vdash M_i \Rightarrow_{\beta\delta} M_i'$. By the Definition Lemma, T is typable in $\Delta_1; x_1{:}A_1, \ldots, x_n{:}A_n$, so by the Free Variable Lemma, $\mathrm{FC}(T) \subseteq \mathrm{DOM}(\Delta_1)$. By Lemma 39, $\Delta_1 \vdash T \Rightarrow_{\beta\delta} T'$. By IH on T, $\Delta_1 \vdash T' \Rightarrow_{\beta\delta} T^{\Delta_1}$. As $\Delta_1 \vdash T \Rightarrow_{\beta\delta} T'$, $\mathrm{FC}(T') \subseteq \mathrm{DOM}(\Delta_1)$, so by Lemma 39, $\Delta \vdash T' \Rightarrow_{\beta\delta} T^{\Delta_1}$. By IH, also $\Delta \vdash M_i' \Rightarrow_{\beta\delta} M_i^\Delta$. Repeatedly applying Lemma 38, we find[13]
      $\Delta \vdash T'[x_1, \ldots, x_n{:=}M_1', \ldots M_n'] \Rightarrow_{\beta\delta}$
      $T^{\Delta_1}[x_1, \ldots, x_n{:=}M_1^\Delta, \ldots, M_n^\Delta]$. ⊠

Corollary 43 (Diamond Property for $\Rightarrow_{\beta\delta}$) *Let Δ be the left part of a context in which M is typable. Assume $\Delta \vdash M \Rightarrow_{\beta\delta} N_1$ and $\Delta \vdash M \Rightarrow_{\beta\delta} N_2$. Then there is P such that $\Delta \vdash N_1 \Rightarrow_{\beta\delta} P$ and $\Delta \vdash N_2 \Rightarrow_{\beta\delta} P$.*

PROOF: Immediately from the theorem above: Take $P \equiv M^\Delta$. ⊠

Theorem 44 (Church-Rosser for $\rightarrow_{\beta\delta}$) *Let Δ be the left part of a context in which M is typable. If $\Delta \vdash M \twoheadrightarrow_{\beta\delta} N_1$ and $\Delta \vdash M \twoheadrightarrow_{\beta\delta} N_2$ then there is P such that $\Delta \vdash N_1 \twoheadrightarrow_{\beta\delta} P$ and $\Delta \vdash N_2 \twoheadrightarrow_{\beta\delta} P$.*

PROOF: Directly from Lemma 37.2, Lemma 37.3 and Corollary 43. ⊠

Lemma 45 (Subject Reduction) *Let $\Delta; \Gamma \vdash A : B$.*
1. If $A \rightarrow_\beta A'$ then $\Delta; \Gamma \vdash A' : B$. *2. $A \rightarrow_\delta A'$ then $\Delta; \Gamma \vdash A' : B$.*
3. If $A \twoheadrightarrow_{\beta\delta} A'$ then $\Delta; \Gamma \vdash A' : B$.

[13] We must remark that $T'[x_1, \ldots, x_n{:=}M_1', \ldots, M_n'] \equiv T'[x_1{:=}M_1'] \cdots [x_n{:=}M_n']$ and $T^{\Delta_1}[x_1, \ldots, x_n{:=}M_1^\Delta, \ldots, M_n^\Delta] \equiv T^{\Delta_1}[x_1{:=}M_1^\Delta] \cdots [x_n{:=}M_n^\Delta]$. This is correct as we can assume that the x_i do not occur in the M_j' and M_j^Δ.

PROOF: The proof for 1. is as in [2]. The proof for 3. is by induction on the length of reduction using 1. and 2. As for 2. we define $\Delta;\Gamma \to_\delta \Delta;\Gamma'$ if $\Gamma \equiv \Gamma_1, x{:}A, \Gamma_2$, and $\Gamma' \equiv \Gamma_1, x{:}A', \Gamma_2$, and $\Delta \vdash A \to_\delta A'$. We define $\Delta;\Gamma \to_\delta \Delta';\Gamma$ similarly. By induction on the derivation of $\Delta;\Gamma \vdash A{:}B$ we prove simultaneously:

$$\begin{aligned}
\Delta;\Gamma \vdash A{:}B \text{ and } \Delta \vdash A \to_\delta A' &\Rightarrow \Delta;\Gamma \vdash A'{:}B\\
\Delta;\Gamma \vdash A{:}B \text{ and } \Delta;\Gamma \to_\delta \Delta';\Gamma &\Rightarrow \Delta';\Gamma \vdash A{:}B\\
\Delta;\Gamma \vdash A{:}B \text{ and } \Delta;\Gamma \to_\delta \Delta;\Gamma' &\Rightarrow \Delta;\Gamma' \vdash A{:}B,
\end{aligned}$$

We only treat the case where the last applied rule is the 2nd application rule, and only prove the first of the three statements. Assume:

$$\Delta \equiv \Delta_1, b{:=}\left(\mathop{\S}_{i=1}^{n} x_i{:}A_i.T\right) : \left(\mathop{\P}_{i=1}^{n} x_i{:}A_i.B\right), \Delta_2 \tag{1}$$

with $B \not\equiv \P y{:}B_1.B_2$, and that the conclusion of the 2nd application rule is

$$\Delta;\Gamma \vdash bM_1 \cdots M_n : K_n \tag{2}$$

for some K_n, and therefore $\Delta \vdash bM_1 \cdots M_n \to_\delta T[x_i{:=}M_i]_{i=1}^n$. We must prove: $\Delta;\Gamma \vdash T[x_i{:=}M_i]_{i=1}^n : K_n$. We do this in two steps.

1. We analyse the structure of K_n, and derive that $\Delta \vdash K_n =_{\beta\delta} B[x_i{:=}M_i]_{i=1}^n$;
2. We show that $\Delta;\Gamma \vdash T[x_i{:=}M_i]_{i=1}^n : B[x_i{:=}M_i]_{i=1}^n$.

Ad 1. We repeatedly apply the Generation Lemma, starting with (2), thus obtaining $K_n, K_{n-1}, \ldots, K_1$, $K'_n, K'_{n-1}, \ldots, K'_1$, $L_n, L_{n-1}, \ldots, L_1$ such that

$$\Delta;\Gamma \vdash bM_1 \cdots M_{i-1} : (\P x_i{:}L_i.K'_i); \tag{3}$$

$$\Delta;\Gamma \vdash M_i : L_i; \tag{4}$$

$$\Delta \vdash K_i =_{\beta\delta} K'_i[x_i{:=}M_i]; \tag{5}$$

$$\Delta \vdash K_{i-1} =_{\beta\delta} \P x_i{:}L_i.K'_i. \tag{6}$$

We end with $\Delta;\Gamma \vdash b : (\P x_1{:}L_1.K'_1)$. By (1) and Generation: $\Delta \vdash \P x_1{:}L_1.K'_1 =_{\beta\delta} \P_{j=1}^n x_j{:}A_j.B$. By Church-Rosser we have $L_1 =_{\beta\delta} A_1$ and

$$\Delta \vdash K'_1 =_{\beta\delta} \mathop{\P}_{j=2}^{n} x_j{:}A_j.B. \tag{7}$$

Hence $\Delta \vdash \P x_2{:}L_2.K'_2 \overset{(6)}{=_{\beta\delta}} K_1 \overset{(5,7)}{=_{\beta\delta}} \left(\P_{j=2}^n x_j{:}A_j.B\right)[x_1{:=}M_1] \equiv \P_{i=2}^n x_i{:}A_i[x_1{:=}M_1].B[x_1{:=}M_1]$, so by the Church-Rosser Theorem we have $L_2 =_{\beta\delta} A_2[x_1{:=}M_1]$. Proceeding in this way, we obtain for $i = 1, \ldots, n$:

$$\begin{aligned}
\Delta \vdash L_i &=_{\beta\delta} A_i[x_j{:=}M_j]_{j=1}^{i-1}; \qquad (8)\\
\Delta \vdash K'_i &=_{\beta\delta} \mathop{\P}_{j=i+1}^{n} x_j{:}A_j[x_k{:=}M_k]_{k=1}^{i-1}.B[x_k{:=}M_k]_{k=1}^{i-1};\\
\Delta \vdash K_i &=_{\beta\delta} \mathop{\P}_{j=i+1}^{n} x_j{:}A_j[x_k{:=}M_k]_{k=1}^{i}.B[x_k{:=}M_k]_{k=1}^{i}.
\end{aligned}$$

In particular,

$$\Delta \vdash K_n =_{\beta\delta} B[x_i{:=}M_i]_{i=1}^n. \tag{9}$$

Ad 2. We calculate the type of $T[x_i{:=}M_i]_{i=1}^n$. By Definition Lemma on (1):

$$\Delta_1; x_1{:}A_1, \ldots, x_n{:}A_n \vdash T : B. \tag{10}$$

By Start Lemma: $\Delta_1; x_1{:}A_1, \ldots, x_{i-1}{:}A_{i-1} \vdash A_i{:}s_i$ for sorts $s_i \in \boldsymbol{S}$. Hence:

$$\begin{array}{rl}
\Delta;\Gamma \vdash A_1 : s_1 & \text{(Thinning Lemma);}\\
\Delta;\Gamma, x_1{:}A_1 \text{ is legal} & \text{(Start Rule);}\\
\Delta;\Gamma, x_1{:}A_1 \vdash A_2 : s_2 & \text{(Thinning Lemma);}\\
\Delta;\Gamma, x_1{:}A_1, x_2{:}A_2 \text{ is legal} & \text{(Start Rule);}\\
\vdots & \\
\Delta;\Gamma, x_1{:}A_1, \ldots, x_n{:}A_n \text{ is legal.} & \text{(Start Rule).}
\end{array}$$

By Thinning Lemma to (10), $\Delta;\Gamma, x_1{:}A_1, \ldots, x_n{:}A_n \vdash T : B$. As $\Delta;\Gamma \vdash M_1 : L_1$ (4) and $\Delta;\Gamma \vdash A_1 : s_1$, we have $\Delta;\Gamma \vdash M_1 : A_1$ by the Conversion rule and (8). By Substitution Lemma: $\Delta;\Gamma, x_2{:}A_2[x_1{:=}M_1], \ldots, x_n{:}A_n[x_1{:=}M_1] \vdash T[x_1{:=}M_1] : B[x_1{:=}M_1]$; and $\Delta;\Gamma \vdash A_2[x_1{:=}M_1] : s_2$.

As $\Delta;\Gamma \vdash M_2 : L_2$ (4) and $\Delta \vdash A_2[x_1{:=}M_1] =_{\beta\delta} L_2$ (8) we have by conversion $\Delta;\Gamma \vdash M_2 : A_2[x_1{:=}M_1]$, and again by the Substitution Lemma:

$$\begin{array}{r}
\Delta;\Gamma, x_3{:}A_3[x_i{:=}M_i]_{i=1}^2, \ldots, x_n{:}A_n[x_i{:=}M_i]_{i=1}^2 \vdash T[x_i{:=}M_i]_{i=1}^2 : B[x_i{:=}M_i]_{i=1}^2\\
\Delta;\Gamma \vdash A_3[x_1{:=}M_1][x_2{:=}M_2] : s_3.
\end{array}$$

Proceeding in this way we eventually find

$$\Delta;\Gamma \vdash T[x_i{:=}M_i]_{i=1}^n : B[x_i{:=}M_i]_{i=1}^n. \tag{11}$$

Applying Lemma 32 to (9) we have $\Delta;\Gamma \vdash K_n : s$. Now use the Conversion Rule, (11), and the fact that $\Delta \vdash K_n =_{\beta\delta} B[x_i{:=}M_i]_{i=1}^n$. ⊠

Lemma 46 *Assume $s \in \boldsymbol{S}$ and M legal. If $\Delta \vdash M =_{\beta\delta} s$ then $M \equiv s$.*

PROOF: First assume $s \in \{\Box, \triangle\}$. If $\Delta;\Gamma \vdash M : N$ for some Γ and N, and $\Delta \vdash M =_{\beta\delta} s$ then by Church-Rosser $\Delta \vdash M \twoheadrightarrow_{\beta\delta} s$, so by Subject Reduction $\Delta;\Gamma \vdash s : N$, contradicting the Generation Lemma. If $\Delta;\Gamma \vdash N : M$ and $\Delta \vdash M =_{\beta\delta} s$ and $M \not\equiv s$ then we have by Lemma 32 that $\Delta;\Gamma \vdash M : P$ for some P, so again $\Delta;\Gamma \vdash s : P$, in contradiction with the Generation Lemma.

Now assume $s \equiv *$, $\Delta;\Gamma \vdash M : N$, and $\Delta \vdash M =_{\beta\delta} s$. By Church-Rosser, $\Delta \vdash M \twoheadrightarrow_{\beta\delta} *$, say $\Delta \vdash M \to_{\beta\delta} \ldots \to_{\beta\delta} M' \to_{\beta\delta} *$. By Subject Reduction, $\Delta;\Gamma \vdash M' : N$ and $\Delta;\Gamma \vdash * : N$. By Generation $\Delta \vdash N =_{\beta\delta} \Box$, so $N \equiv \Box$.

- $M' \equiv (\lambda x{:}A.B)C$ and $* \equiv B[x{:=}C]$. By Generation $\exists B'$ where $\Delta \vdash B'[x{:=}C] =_{\beta\delta} \Box$ (so $B'[x{:=}C] \equiv \Box$), $\Delta;\Gamma \vdash (\lambda x{:}A.B) : (\Pi x{:}A.B')$ and $\Delta;\Gamma \vdash C : A$. $C \equiv \Box$ contradicts $\Delta;\Gamma \vdash C : A$, so $B' \equiv \Box$. By Lemma 32, $\Delta;\Gamma \vdash (\Pi x{:}A.\Box) : *$, so by Generation $\Delta;\Gamma, x{:}A \vdash \Box : *$, contradiction;
- $M' \equiv bM_1 \cdots M_n$ and $\Delta \vdash bM_1 \cdots M_n \to_\delta T[x_i{:=}M_i]_{i=1}^n \equiv *$ as above.

If $s \equiv *$, $\Delta;\Gamma \vdash N : M$, and $\Delta \vdash M =_{\beta\delta} s$ then by Lemma 32 $M \equiv s$ (and we are done) or $\Delta;\Gamma \vdash M : s'$ (which implies $M \equiv s$ by the above argument). $\boxtimes$

We prove Strong Normalisation for $\beta\delta$-reduction in $\lambda 68$ by mapping a typable term M (in a context $\Delta;\Gamma$) of $\lambda 68$ to a term $|M|_\Delta$ that is typable in a strongly normalising PTS. The mapping is constructed in such a way that if $M \to_\beta N$, $|M|_\Delta \twoheadrightarrow_\beta^+ |N|_\Delta$, and that if $\Delta \vdash M \to_\delta N$, $|M|_\Delta \twoheadrightarrow_\beta |N|_\Delta$.

Definition 47 Let Δ be the left part of a legal context and let $M \in \mathcal{T}$. We define $|M|_\Delta$ by induction on the length of Δ and the structure of M.

- $|x|_\Delta \stackrel{\text{def}}{=} x$ for $x \in \mathcal{V}$;
- $|s|_\Delta \stackrel{\text{def}}{=} s$ for $s \in \boldsymbol{S}$
- $|PQ|_\Delta \stackrel{\text{def}}{=} |P|_\Delta\,|Q|_\Delta$
- $|\lambda x{:}P.Q|_\Delta \stackrel{\text{def}}{=} \lambda x{:}\,|P|_\Delta\,.\,|Q|_\Delta$
- $|\Pi x{:}P.Q|_\Delta \stackrel{\text{def}}{=} \Pi x{:}\,|P|_\Delta\,.\,|Q|_\Delta$
- $|\P x{:}P.Q|_\Delta \stackrel{\text{def}}{=} \Pi x{:}\,|P|_\Delta\,.\,|Q|_\Delta$
- $|b|_\Delta \stackrel{\text{def}}{=} b$ for all $b \in \mathcal{C} \setminus \textsc{defcons}(\Delta;)$
- $|b|_\Delta \stackrel{\text{def}}{=} \lambda_{i=1}^n x_i{:}\,|A_i|_{\Delta_1}\,.\,|T|_{\Delta_1}$ if $\Delta \equiv \Delta_1, b{:=}(\S_{i=1}^n x_i{:}A_i.T){:}(\P_{i=1}^n x_i{:}A_i.U), \Delta_2$

The following lemma is useful:

Lemma 48 *Let Δ, Δ_1, Δ_2 be left parts of legal contexts and $M, N \in \mathcal{T}$.*

1. $\textsc{fv}(|M|_\Delta) = \textsc{fv}(M)$.
2. *If $\Delta_2 \equiv \Delta_1, \Delta'$ and $\textsc{fc}(M) \subseteq \textsc{dom}(\Delta_1)$ then $|M|_{\Delta_2} \equiv |M|_{\Delta_1}$.*
3. $|M[x{:=}N]|_\Delta \equiv |M|_\Delta\,[x{:=}|N|_\Delta]$.

Proof: 1. is by induction on the definition of $|M|_\Delta$. We show the non trivial case where $M \equiv b$ and $\Delta \equiv \Delta_1, b{:=}(\S\,\Gamma.T){:}(\P\,\Gamma.U), \Delta_2$ ($T \not\equiv \S\,y{:}T_1.T_2$). By the Definition Lemma, T is typable in $\Delta_1;\Gamma$; therefore $\textsc{fv}(T) \subseteq \textsc{dom}(\Gamma)$ (Free Variable Lemma). By IH, $\textsc{fv}\big(|T|_{\Delta_1}\big) \subseteq \textsc{dom}(\Gamma)$ and therefore $\textsc{fv}(|b|_\Delta) = \varnothing$.

2. is by an easy induction on the definition of $|M|_{\Delta_1}$.

3. is by induction on the definition of $|M|_\Delta$. In the case $M \equiv b$ and $b{:=}T{:}U \in \Delta$, use the fact that $\textsc{fv}(|M|_\Delta) = \textsc{fv}(M) = \varnothing$ (Lemma48.1) and therefore $|M|_\Delta\,[x{:=}|N|_\Delta] \equiv |M|_\Delta \equiv |M[x{:=}N]|_\Delta$. $\boxtimes$

The purpose of the definition of $|M|_\Delta$ is explained in the following lemma:

Lemma 49 *1. If $M \to_\beta N$ then $|M|_\Delta \twoheadrightarrow_\beta^+ |N|_\Delta$.*

2. If $\Delta \vdash M \to_\delta N$, then $|M|_\Delta \twoheadrightarrow_\beta |N|_\Delta$.

PROOF: 1. is by induction on the structure of M. We only treat the case $M \equiv (\lambda x{:}P.Q)R$ and $N \equiv Q[x{:=}R]$. Then

$|M|_\Delta \equiv (\lambda x{:}\,|P|_\Delta \,.\, |Q|_\Delta)\,|R|_\Delta \to_\beta |Q|_\Delta\,[x{:=}\,|R|_\Delta] \overset{48.3}{\equiv} |Q[x{:=}R]|_\Delta$.

2. is by induction on the structure of M. We only treat the case in which $M \equiv bM_1\cdots M_n$; $\Delta \equiv \Delta_1, b{:=}(\S_{i=1}^n\, x_i{:}A_i.T) : (\P_{i=1}^n\, x_i{:}A_i.U)\,, \Delta_2$; and $N \equiv T[x_1,\ldots,x_n{:=}M_1,\ldots,M_n]$. Note that

$|M|_\Delta \equiv \left(\lambda_{i=1}^n\, x_i{:}\,|A_i|_{\Delta_1}\,.\,|T|_{\Delta_1}\right)|M_1|_\Delta\cdots|M_n|_\Delta \twoheadrightarrow_\beta |T|_{\Delta_1}\,[x_i{:=}\,|M_i|_\Delta]_{i=1}^n$
$\overset{48.2}{\equiv} |T|_\Delta\,[x_i{:=}\,|M_i|_\Delta]_{i=1}^n \overset{48.3}{\equiv} |T[x_i{:=}M_i]_{i=1}^n|_\Delta \equiv |T[x_1,\ldots,x_n{:=}M_1,\ldots,M_n]|_\Delta$.

At the last equivalence, we must make a remark similar to footnote 13. ⊠

Let λSN be the PTS over λ-terms with variables from $\mathcal{V}\cup\mathcal{C}$ and sorts from $\boldsymbol{S}$, and the rules:[14] $(*,*,*)$; $(*,*,\triangle)$; $(\Box,*,\triangle)$; $(*,\Box,\triangle)$; $(\Box,\Box,\triangle)$; $(*,\triangle,\triangle)$; $(\Box,\triangle,\triangle)$.

This is in fact the pure type system that is based on the Π-formation rules of Section 3.1. λSN is contained in **ECC** [2]. As **ECC** is β-strongly normalising, also λSN is β-strongly normalising.

We present a translation of λ68-contexts to λSN-contexts:

Definition 50 Let $\Delta;\Gamma$ be a legal λ68-context.

- We define $|\Delta|$ by induction on the length of Δ:
  - $|\varnothing| \overset{\text{def}}{=} \varnothing$; • $|\Delta, b{:}U| \overset{\text{def}}{=} |\Delta|\,, b{:}\,|U|_\Delta$; • $|\Delta, b{:=}T{:}U| \overset{\text{def}}{=} |\Delta|$;
- If $\Gamma \equiv x_1{:}A_1,\ldots,x_n{:}A_n$ then $|\Delta;\Gamma| \overset{\text{def}}{=} |\Delta|\,, x_1{:}\,|A_1|_\Delta\,,\ldots,x_n{:}\,|A_n|_\Delta$.

We see that definitions $b{:=}T{:}U$ in Δ are not translated into $|\Delta|$. This corresponds to the fact that all these definitions are unfolded (replaced by their definiendum) in $|b|_\Delta$. Now we prove a very important lemma:

Lemma 51 *If* $\Delta;\Gamma \vdash_{\lambda 68} M : N$ *then* $|\Delta;\Gamma| \vdash_{\lambda\text{SN}} |M|_\Delta : |N|_\Delta$.

PROOF: By induction on the derivation of $\Delta;\Gamma \vdash M : N$. We treat the cases:

(Start: Primitive Constants) $\dfrac{\Delta;\Gamma \vdash_{\lambda 68} B : s_1 \;\; \Delta;\vdash_{\lambda 68} \P\,\Gamma.B : s_2}{\Delta, b{:}\,\P\,\Gamma.B;\vdash_{\lambda 68} b : \P\,\Gamma.B}\, s_1 = *,\Box$

By IH, $|\Delta| \vdash_{\lambda\text{SN}} |\P\,\Gamma.B|_\Delta : s_2$, so by Start $|\Delta|\,, b{:}\,|\P\,\Gamma.B|_\Delta \vdash b{:}\,|\P\,\Gamma.B|_\Delta$. Observe that $|\Delta, b{:}\,\P\,\Gamma.B| \equiv |\Delta|\,, b{:}\,|\P\,\Gamma.B|_\Delta$, that $|b|_{\Delta, b:\P\,\Gamma.B} \equiv b$ and that (by Lemma 48.2) $|\P\,\Gamma.B|_\Delta \equiv |\P\,\Gamma.B|_{\Delta, b:\P\,\Gamma.B}$;

(Start: Defined Constants) $\dfrac{\Delta;\Gamma \vdash_{\lambda 68} T : B : s_1 \;\; \Delta;\vdash_{\lambda 68} \P\,\Gamma.B : s_2}{\Delta, b{:=}(\Gamma.T){:}(\P\,\Gamma.B);\vdash_{\lambda 68} b : \P\,\Gamma.B}\, s_1 = *,\Box$

By induction $|\Delta;| \vdash_{\lambda\text{SN}} |\P\,\Gamma.B|_\Delta : s_2$, so (write $\Gamma \equiv x_1{:}A_1,\ldots,x_n{:}A_n$):

$$|\Delta;| \vdash_{\lambda\text{SN}} \prod_{i=1}^n x_i{:}\,|A_i|_\Delta\,.\,|B|_\Delta : s_2. \tag{12}$$

[14] We choose the name λSN because this system will help us in showing that λ68 is SN.

By induction, we also have $|\Delta;\Gamma| \vdash_{\lambda\mathrm{SN}} |T|_\Delta : |B|_\Delta$, so:

$$|\Delta|\,, x_1{:}\,|A_1|_\Delta\,, \ldots, x_n{:}\,|A_n|_\Delta \vdash_{\lambda\mathrm{SN}} |T|_\Delta : |B|_\Delta\,, \tag{13}$$

and by repeatedly applying the λ-rule on (13) and using the fact that, by IH, the types $\prod_{j=i}^{n} x_j{:}\,|A_j|_\Delta\,.\,|B|_\Delta$ are all typable, we find:

$$|\Delta;| \vdash_{\lambda\mathrm{SN}} \left(\underset{i=1}{\overset{n}{\lambda}} x_i{:}\,|A_i|_\Delta\,.\,|T|_\Delta\right) : \left(\prod_{i=1}^{n} x_i{:}\,|A_i|_\Delta\,.\,|B|_\Delta\right); \tag{14}$$

(Application 1) $\dfrac{\Delta;\Gamma \vdash_{\lambda 68} M : (\Pi x{:}A.B) \qquad \Delta;\Gamma \vdash_{\lambda 68} N : A}{\Delta;\Gamma \vdash_{\lambda 68} MN : B[x{:=}N]}$.

By IH, $|\Delta;\Gamma| \vdash_{\lambda\mathrm{SN}} |M|_\Delta : (\Pi x{:}\,|A|_\Delta\,.\,|B|_\Delta)$ and $|\Delta;\Gamma| \vdash_{\lambda\mathrm{SN}} |N|_\Delta : |A|_\Delta$. By application $|\Delta;\Gamma| \vdash_{\lambda\mathrm{SN}} |M|_\Delta\,|N|_\Delta : |B|_\Delta\,[x{:=}|A|_\Delta]$. By definition of $|MN|_\Delta$ and Lemma 48.3, $|\Delta;\Gamma| \vdash_{\lambda\mathrm{SN}} |MN|_\Delta : |B[x{:=}A]|_\Delta$. ⊠

Theorem 52 (Strong Normalisation) *$\lambda 68$ is $\beta\delta$-strongly normalising.*

Proof: Assume, we have an infinite $\beta\delta$-reduction path in $\lambda 68$:

$$M_1 \to_{\beta\delta} M_2 \to_{\beta\delta} M_3 \to_{\beta\delta} \ldots \tag{15}$$

As δ-reduction is strongly normalising (Lemmas 20 and 35), there must be infinitely many β-reductions in this reduction path, so we have a path $N_1 \to_\beta N_1' \twoheadrightarrow_\delta N_2 \to_\beta N_2' \twoheadrightarrow_\delta N_3 \to_\beta N_3' \twoheadrightarrow_\delta \ldots$ By Lemmas 49.1 and 49.2, this gives a path $|N_1|_\Delta \twoheadrightarrow_\beta^+ |N_1'|_\Delta \twoheadrightarrow_\beta |N_2|_\Delta \twoheadrightarrow_\beta^+ |N_2'|_\Delta \twoheadrightarrow_\beta |N_3|_\Delta \twoheadrightarrow_\beta^+ |N_3'|_\Delta \twoheadrightarrow_\beta \ldots$ which is an infinite β-reduction path in λSN. By Lemma 51, $|N_1|_\Delta$ is legal in λSN. But as λSN is strongly normalising, this infinite β-reduction path cannot exist. Hence, the infinite $\beta\delta$-reduction path (15) does not exist, either. ⊠

The next two theorems establish the formal relation between Aut-68 and $\lambda 68$.

Theorem 53 *Let $\mathfrak{B}$ be an* Automath *book and Γ an* Automath *context.*

- *If $\mathfrak{B};\Gamma \vdash_{AUT-68}$ OK then $\overline{\mathfrak{B}};\overline{\Gamma}$ is legal;*
- *If $\mathfrak{B};\Gamma \vdash_{AUT-68} \Sigma : \Omega$ then $\overline{\mathfrak{B}};\overline{\Gamma} \vdash_{\lambda 68} \overline{\Sigma} : \overline{\Omega}$.*

Proof: We prove both statements simultaneously, by induction on the derivation of $\mathfrak{B};\Gamma \vdash_{\mathrm{AUT-68}}$ OK and $\mathfrak{B};\Gamma \vdash \Sigma : \Omega$ of Definitions 14 and 15. We only treat one case. Assume, the last step is book extension rule def2:

$$\frac{\mathfrak{B};\Gamma \vdash_{\mathrm{AUT-68}} \Sigma_2{:}\mathtt{type} \qquad \mathfrak{B};\Gamma \vdash_{\mathrm{AUT-68}} \Sigma_1{:}\Sigma_2' \qquad \mathfrak{B};\Gamma \vdash_{\mathrm{AUT-68}} \Sigma_2 =_{\beta\mathrm{d}} \Sigma_2'}{\mathfrak{B},(\Gamma;k;\Sigma_1;\Sigma_2);\varnothing \vdash_{\mathrm{AUT-68}} \mathrm{OK}}.$$

By IH, we have

$$\overline{\mathfrak{B}};\overline{\Gamma} \vdash_{\lambda 68} \overline{\Sigma_2} : *\tag{16}$$

and

$$\overline{\mathfrak{B}};\overline{\Gamma} \vdash_{\lambda 68} \overline{\Sigma_1} : \overline{\Sigma_2'}.\tag{17}$$

By Lemma 35, we have

$$\overline{\mathfrak{B}} \vdash_{\lambda 68} \overline{\Sigma_2} =_{\beta\delta} \overline{\Sigma_2'}. \tag{18}$$

Applying the conversion rule of λ68 to (16), (17) and (18) yields

$$\overline{\mathfrak{B}}; \overline{\Gamma} \vdash_{\lambda 68} \overline{\Sigma_1} : \overline{\Sigma_2}. \tag{19}$$

As $\overline{\mathfrak{B}}; \overline{\Gamma}$ is legal, for each $x{:}\alpha \in \overline{\Gamma}$ (say: $\overline{\Gamma} \equiv \Gamma_1, x{:}\alpha, \Gamma_2$) we have $\overline{\mathfrak{B}}; \Gamma_1 \vdash \alpha : s$ for an $s \in \{*, \Box\}$, by the Free Variable Lemma 26. Thus we can repeatedly apply the ¶-formation rule (starting with (16)) to obtain:

$$\overline{\mathfrak{B}}; \vdash_{\lambda 68} \P\, \overline{\Gamma}.\overline{\Sigma_2} : \triangle \tag{20}$$

(If $\Gamma \equiv \varnothing$ then we apply the ¶-formation rule zero times, and the type of $\P\, \overline{\Gamma}.\overline{\Sigma_2}$ is $*$ instead of $\triangle$). Now we can apply the (Start: dc) rule on (19), (16) and (20) to obtain: $\overline{\mathfrak{B}}; k{:=}(\S\, \overline{\Gamma}.\overline{\Sigma_1}){:}(\P\, \overline{\Gamma}.\overline{\Sigma_2}); \vdash_{\lambda 68} k : \P\, \overline{\Gamma}.\overline{\Sigma_2}$,
so $\overline{\mathfrak{B}, (\Gamma; k; \Sigma_1; \Sigma_2);} \equiv \overline{\mathfrak{B}}, k{:=}(\S\, \overline{\Gamma}.\overline{\Sigma_1}){:}(\P\, \overline{\Gamma}.\overline{\Sigma_2});$ is legal. ⊠

Theorem 54 *Let $\Delta; \Gamma \vdash_{\lambda 68} M : N$. There is an* AUTOMATH *book $\mathfrak{B}$ and context Γ' such that $\mathfrak{B}; \Gamma' \vdash_{AUT\text{-}68}$ OK, and $\overline{\mathfrak{B}}, \overline{\Gamma'} \equiv \Delta; \Gamma$. Moreover,*

1. *If $N \equiv \Box$ then $M \equiv *$;*
2. *If $\Delta; \Gamma \vdash_{\lambda 68} N : \Box$ then $N \equiv *$ and there is $\Omega \in \mathcal{E}$ such that $\overline{\Omega} \equiv M$ and $\mathfrak{B}; \Gamma' \vdash_{AUT\text{-}68} \Omega :$ `type`;*
3. *If $N \equiv \triangle$ then there is $\Gamma'' \equiv x_1{:}\Sigma_1, \ldots, x_n{:}\Sigma_n$ and $\Omega \in \mathcal{E} \cup \{$`type`$\}$ such that: • Γ', Γ'' is correct with respect to $\mathfrak{B}$; • $M \equiv \P\, \overline{\Gamma''}.\overline{\Omega}$;*
   • $\Omega \equiv$ `type` or $\mathfrak{B}; \Gamma' \vdash_{AUT\text{-}68} \Omega :$ `type`;
4. *If $\Delta; \Gamma \vdash_{\lambda 68} N : \triangle$ then there are $b \in \mathcal{C}$ and $\Sigma_1, \ldots, \Sigma_n \in \mathcal{E}$ such that $M \equiv b\overline{\Sigma_1} \cdots \overline{\Sigma_n}$. Moreover, $\mathfrak{B}$ contains a line $(x_1{:}\Omega_1, \ldots, x_m{:}\Omega_m; b; \Xi_1; \Xi_2)$ such that: • $N \equiv \left(\P_{i=n+1}^{m}\, x_i{:}\overline{\Omega_i}.\overline{\Xi_2}\right)[x_1, \ldots, x_n{:=}\overline{\Sigma_1}, \ldots, \overline{\Sigma_n}]$; • $m > n$;*
   • $\mathfrak{B}; \Gamma' \vdash_{AUT\text{-}68} \Sigma_i{:}\Omega_i[x_1, \ldots, x_{i-1}{:=}\Sigma_1, \ldots, \Sigma_{i-1}]$ $(1 \leq i \leq n)$;
5. *If $N \equiv *$ then $\exists \Omega \in \mathcal{E}$ such that $\overline{\Omega} \equiv M$ and $\mathfrak{B}; \Gamma' \vdash_{AUT\text{-}68} \Omega :$ `type`;*
6. *If $\Delta; \Gamma \vdash_{\lambda 68} N : *$ then there are $\Sigma, \Omega \in \mathcal{E}$ such that $\overline{\Sigma} \equiv M$ and $\overline{\Omega} \equiv N$, and $\mathfrak{B}; \Gamma' \vdash_{AUT\text{-}68} \Sigma : \Omega$, and $\mathfrak{B}; \Gamma' \vdash_{AUT\text{-}68} \Omega :$ `type`.*

PROOF: Induction on the derivation of $\Delta; \Gamma \vdash_{\lambda 68} M : N$. We treat the cases:

Weakening: definitions Assume the last step is

$$\frac{\Delta; \vdash_{\lambda 68} M : N \qquad \Delta; \Gamma \vdash_{\lambda 68} T : B : s_1 \qquad \Delta; \vdash_{\lambda 68} \P\, \Gamma.B : s_2}{\Delta, b{:=}(\S\, \Gamma.T){:}(\P\, \Gamma.B); \vdash_{\lambda 68} M : N}$$ where $s_1 \equiv *$ or $s_1 \equiv \Box$. Use IH and determine $\mathfrak{B}$, Γ', Σ_1, Σ_2, Ω_1, and Ω_2 such that $\overline{\mathfrak{B}} \equiv \Delta$, $\overline{\Gamma'} \equiv \Gamma$, $\overline{\Sigma_1} \equiv T$, $\overline{\Sigma_2} \equiv B$, $\overline{\Omega_1} \equiv M$ and $\overline{\Omega_2} \equiv N$. We know by induction that $\mathfrak{B}; \Gamma' \vdash_{\mathrm{AUT}-68} \Sigma_2 :$ `type` (if $s_1 \equiv *$) or $\Sigma_2 \equiv *$ (if $s_2 \equiv \Box$). Also, $\mathfrak{B}; \Gamma' \vdash_{\mathrm{AUT}-68} \Sigma_1 : \Sigma_2$. This makes it possible to extend $\mathfrak{B}$ with a new line, thus obtaining a legal book $\mathfrak{B}, (\Gamma'; b; \Sigma_1; \Sigma_2)$. Using Weakening for AUT-68 (Lemma 22) and IH on $\Delta; \vdash_{\lambda 68} M : N$, it is not hard to verify the cases 1–6 for $\Delta, b{:=}(\S\, \Gamma.T){:}(\P\, \Gamma.B); \vdash_{\lambda 68} M : N$;

Application 2 The last step is $\dfrac{\Delta;\Gamma \vdash_{\lambda 68} M_1 : (\P x{:}A.B) \qquad \Delta;\Gamma \vdash_{\lambda 68} M_2 : A}{\Delta;\Gamma \vdash_{\lambda 68} M_1 M_2 : B[x{:=}M_2]}$.

Determine $\mathfrak{B}$, Γ' such that $\overline{\mathfrak{B}} \equiv \Delta$ and $\overline{\Gamma'} \equiv \Gamma$. By Correctness of Types 32 and Generation Lemma 31, $\Delta;\Gamma \vdash_{\lambda 68} (\P x{:}A.B) : \triangle$, so by IH (case 4), there are $b, \Sigma_1, \ldots, \Sigma_n$ such that $M_1 \equiv b\overline{\Sigma_1}\cdots\overline{\Sigma_n}$, and there is a line $(x_1{:}\Omega_1, \ldots, x_m{:}\Omega_m; b; \Xi_1; \Xi_2)$ in $\mathfrak{B}$ such that $m > n$, $\mathfrak{B};\Gamma' \vdash_{\mathrm{AUT}-68} \Sigma_i{:}\Omega_i[x_j{:=}\Sigma_j]_{j=1}^{i-1}$ $1 \leq i \leq n$ and $\P x{:}A.B \equiv \left(\P_{i=n+1}^{m} x_i{:}\overline{\Omega_i}.\overline{\Xi_2}\right)[x_j{:=}\overline{\Sigma_j}]_{j=1}^{n}$. Observe: $A \equiv \overline{\Omega_{n+1}}[x_j{:=}\overline{\Sigma_j}]_{j=1}^{n}$. As $\mathfrak{B};\Gamma' \vdash_{\mathrm{AUT}-68} \Omega_{n+1} : \mathtt{type}$ or $\Omega_{n+1} \equiv \mathtt{type}$, we have $\Delta;\Gamma \vdash_{\lambda 68} \overline{\Omega_{n+1}} : s$ for an $s \in \{*, \Box\}$, and by Substitution and Transitivity Lemmas we have $\Delta;\Gamma \vdash_{\lambda 68} \overline{\Omega_{n+1}}[x_j{:=}\overline{\Sigma_j}]_{j=1}^{n} : s$, hence $\Delta;\Gamma \vdash_{\lambda 68} A : s$. With IH we determine $\Sigma \in \mathcal{E}$ such that $\mathfrak{B};\Gamma' \vdash_{\mathrm{AUT}-68} \Sigma : \Omega_{n+1}[x_j{:=}\Sigma_j]_{j=1}^{n}$, and $M_2 \equiv \overline{\Sigma}$.

We now treat the most important ones of the cases 1–6:

4. The only thing that does not directly follow from the results above is $m > n+1$. Assume, for the sake of the argument, $m = n+1$. Then $B[x{:=}M_2] \equiv \overline{\Xi_2}[x_j{:=}\overline{\Sigma_j}]_{j=1}^{n+1}$. As $\Delta;\Gamma \vdash_{\lambda 68} B[x{:=}M_2] : \triangle$, $\overline{\Xi_2}[x_j{:=}\overline{\Sigma_j}]_{j=1}^{n+1}$ is of the form $\P x{:}P.Q$, which is impossible;
6. Note: $B[x{:=}M_2] \equiv \left(\P_{j=n+2}^{m} x_i{:}\overline{\Omega_i}.\overline{\Xi_2}\right)[x_j{:=}\overline{\Sigma_j}]_{j=1}^{n+1}$. We have $\Delta;\Gamma \vdash_{\lambda 68} B[x{:=}M_2] : *$. So $B[x{:=}M_2] \not\equiv \P y{:}P.Q$, and hence $m = n+1$. Therefore, $\mathfrak{B};\Gamma' \vdash_{\mathrm{AUT}-68} b(\Sigma_1, \ldots, \Sigma_{n+1}) : \Xi_2[x_i{:=}\Sigma_i]_{i=1}^{n+1}$. ⊠

Remark 55 We explain different cases used in the formulation of Theorem 54.

- The cases $N \equiv \Box$ and $\Delta;\Gamma \vdash N : \Box$ imply that there are no other terms in $\lambda 68$ than $*$ itself at the same level as $*$. This corresponds to the fact that `type` is the only "top-expression" in AUT-68;
- The cases $N \equiv *$ and $\Delta;\Gamma \vdash N : *$ give a precise correspondence between expressions of AUT-68 and terms of $\lambda 68$: If $M : N$ in $\lambda 68$ then there are Σ, Ω in AUT-68 such that $\Sigma : \Omega$ in AUT-68 and $\overline{\Sigma} \equiv M$ and $\overline{\Omega} \equiv N$;
- The cases $N \equiv \triangle$ and $\Delta;\Gamma \vdash N : \triangle$ cover terms that do not have an equivalent in AUT-68 but are necessary in $\lambda 68$ to form terms that have equivalents in AUT-68. More specific, this concerns terms of the form $\P_{i=1}^{n} x_i{:}A_i.B$ (needed to introduce constants) and terms of the form $bM_1 \cdots M_n$, where b is a constant of type $\P_{i=1}^{m} x_i{:}A_i.B$ for certain $m > n$ (needed to construct $\lambda 68$-equivalents of expressions like $b(\Sigma_1, \ldots, \Sigma_m)$).

We conclude that $\lambda 68$ and AUT-68 coincide as much as possible, and that the terms in $\lambda 68$ that do not have an equivalent in AUT-68 can be traced easily (these are the terms of type $\triangle$ and the terms of a type $N : \triangle$, and the sorts $\Box$ and $\triangle$, which are needed to give a type to $*$ and to the $\P$-types).

Notice that the alternative definition of δ-reduction in $\lambda 68$, discussed at the end of Subsection 2.5, would introduce more terms in $\lambda 68$ without an equivalent in AUT-68, namely terms of the form $\lambda_{i=1}^{n} x_i{:}A_i.B$.

5 More suitable PTSs for AUTOMATH systems

Recall that we related the system AUT-68 to a PTS $\lambda 68$ ignoring the AUTOMATH features: parameters, and identifying λs and Πs or at least, providing both Π-reduction and Π-application. In particular, in Definition 23, we gave $\overline{b(\Sigma_1,\ldots,\Sigma_n)} \stackrel{\text{def}}{=} b\overline{\Sigma_1}\cdots\overline{\Sigma_n}$ as $\lambda 68$ does not have direct parameters. Also, although we had λs and Πs in $\lambda 68$, unlike AUTOMATH which used expressions of the form $[x{:}\Sigma]\Omega$ for both abstractions, we did not allow neither Π-reduction where the reduction rule $\to_\Pi$ works like β-reduction as follows:

Π-reduction $\quad (\Pi x{:}A.B)N \to_\Pi B[x{:=}N]$

nor Π-application where the $\lambda 68$ rule (App_1) is changed into

$$\textbf{Π-application} \qquad \frac{\Delta;\Gamma \vdash M : \Pi x{:}A.B \qquad \Delta;\Gamma \vdash N : A}{\Delta;\Gamma \vdash MN : (\Pi x{:}A.B)N}$$

There are good reasons to use parameters (cf. [32, 33]), Π-reduction and Π-application (cf. [31, 36]). In Section 5.1 we look at how we might remedy the above shorcomings to create more faithful interpretations of AUT-68 as PTSs.

The system AUT-68 is one of several AUTOMATH-systems that have been proposed. Another frequently used system is AUT-QE. In Section 5.2 we compare AUT-68 to AUT-QE and describe how we can easily adapt $\lambda 68$ to a system λQE. In Section 5.3 we reflect on the system $\Delta\Lambda$ which is claimed by de Bruijn to embrace all the essential aspects of AUTOMATH apart from type inclusion.

5.1 $\lambda 68$ with parameters, Π-reduction and Π-application

PTSs don't usually follow AUTOMATH in identifying λs and Πs. PTSs don't even follow AUTOMATH in allowing Π-reduction and Π-application. We have the following results in the area:

- [30] showed that as long as the usual application rule of PTSs is used, a PTS system remains unchanged whether Π-reduction is included or not. As a result, if the usual application rule of PTSs is used, a PTS system remains unchanged whether λs and Πs are unified or not. [30] concluded that a PTS system where λs and Πs are unified and where the application is changed to Π-application faces the same problem (and inherits the same solution) as that of the PTSs where λs and Πs are not unified but where Π-application and Π-reduction are used.

- [36] showed that PTSs with Π-reduction and Π-application lose Subject Reduction. For instance, one can derive $\alpha{:}*, x{:}\alpha \vdash (\lambda y{:}\alpha.y)x : (\Pi y{:}\alpha.\alpha)x$, but it is not possible to derive $\alpha{:}*, x{:}\alpha \vdash x : (\Pi y{:}\alpha.\alpha)x$.

- [31] showed that PTSs with Π-reduction and Π-application have all the desirable properties if a definition system is used. Let us call the PTS with Π-reduction and Π-application and definitions as in [31], $\lambda\beta\Pi$d.

Though our system $\lambda 68$ does not have Π-reduction and Π-application, it is easy to extend it to a system $\lambda\Pi 68$ by adding these rules:

- Changing rule (App_1) into $\dfrac{\Delta;\Gamma \vdash M : \Pi x{:}A.B \qquad \Delta;\Gamma \vdash N : A}{\Delta;\Gamma \vdash MN : (\Pi x{:}A.B)N}$

  (Rule (App_2) remains unchanged — see also the discussion in Section 3.1);

- Adding the new reduction rule $\to_\Pi$ by $(\Pi x{:}A.B)N \to_\Pi B[x{:=}N]$.

The system $\lambda\Pi 68$ is actually much closer to AUT-68 than $\lambda 68$.

In $\lambda\Pi 68$ we do not have Subject Reduction, either: it is not hard to derive

$$;\alpha{:}*, x{:}\alpha \vdash (\lambda y{:}\alpha.y)x : (\Pi y{:}\alpha.\alpha)x$$

Nevertheless, we can not derive in $\lambda\Pi 68$

$$;\alpha{:}*, x{:}\alpha \vdash x : (\Pi y{:}\alpha.\alpha)x$$

The "restoration" of Subject Reduction in $\lambda\beta\Pi\mathrm{d}$ is only because of the special way in which definitions are introduced and removed from the context. In $\lambda\Pi 68$, once definitions have been introduced, they cannot be removed from the left part of the context any more. So, we need to investigate whether the method of [31] can be extended to $\lambda\Pi 68$ in order to restore Subject Reduction in $\lambda\Pi 68$.

As for parameters, [32] gives a formulation of PTSs with parameters, [33] formulates PTSs with parameters, Π-reduction, Π-application, definitions à la [31] and explicit substitutions, [41, 6] formulate PTSs with parameters and definitions as in AUTOMATH and [30] gives a formulation of PTSs where λs and Πs are unified, and with parameters, Π-application, explicit substitutions and definitions à la [31]. All these formulations satisfy the good properties of PTSs.

In the above systems, PTSs are extended with parameters by adding terms of the form $\mathcal{C}(A_1, \ldots, A_n)$ where $\mathcal{C}$ is a set of constants disjoint from the set of variables, and $n \geq 0$. Then, in addition to the set of (Π-formation) rules $\boldsymbol{R}$, a set of parametric construction rules $\boldsymbol{P}$ is added. Typing rules for dealing with the new terms are finally added as follows: ($\Delta \equiv x_1{:}B_1, \ldots, x_n{:}B_n$, $\Delta_i \equiv x_1{:}B_1, \ldots, x_{i-1}{:}B_{i-1}$ and $\mathrm{CONS}(\Gamma)$ is the set of constant declarations in Γ):

$$\textbf{($\vec{\mathbf{C}}$-weak)} \quad \frac{\Gamma \vdash_a C : B \quad \Gamma, \Delta_i \vdash_a B_i : s_i \quad \Gamma, \Delta \vdash_a A : s}{\Gamma, c(\Delta) : A \vdash_a C : B}(s_i, s) \in \boldsymbol{P},\ c \notin \mathrm{CONS}(\Gamma)$$

$$\textbf{($\vec{\mathbf{C}}$-app)} \quad \frac{\begin{array}{lll} \Gamma_1, c(\Delta){:}A, \Gamma_2 & \vdash_a & A_i{:}B_i[x_j{:=}A_j]_{j=1}^{i-1} \qquad (i = 1, \ldots, n) \\ \Gamma_1, c(\Delta){:}A, \Gamma_2 & \vdash_a & A : s \qquad\qquad\qquad\qquad\quad (\text{if } n = 0) \end{array}}{\Gamma_1, c(\Delta){:}A, \Gamma_2 \vdash_a c(A_1, \ldots, A_n) : A[x_j{:=}A_j]_{j=1}^{n}}$$

With this in mind, the Barendregt cube of Figure 2 can be refined into the eight smaller cubes on the left, and the AUTOMATH systems AUT-68 and AUT-QE, as well as the Edinburgh LF and Milner's ML find a more accurate placing in this refined cube as on the picture on the right (cf. [32, 33, 41]).

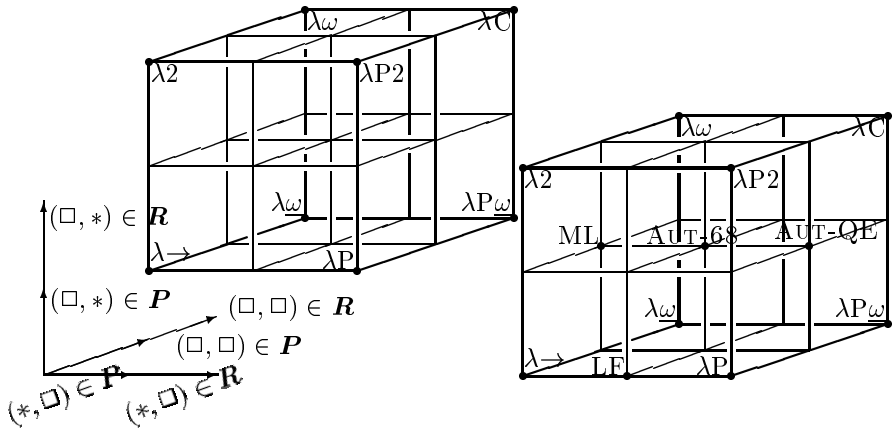

Figure 6: LF, ML, AUT-68, and AUT-QE in the refined Barendregt Cube

5.2 AUT-QE

The system AUT-QE has many similarities with AUT-68, and a few extensions:

1. We can form abstraction expression $[x{:}\Sigma]\mathtt{type}$ (extending Definition 8);
2. Inhabitants of types of the form $[x{:}\Sigma]\mathtt{type}$ are introduced by extending the abstraction rules 1 and 2 of Definition 15 with the rule for AUT-QE:

$$\frac{\mathfrak{B};\Gamma \vdash \Sigma_1{:}\mathtt{type} \qquad \mathfrak{B};\Gamma, x{:}\Sigma_1 \vdash \Sigma_2{:}\mathtt{type}}{\mathfrak{B};\Gamma \vdash [x{:}\Sigma_1]\Sigma_2 : [x{:}\Sigma_1]\mathtt{type}}.$$

   Notice that the expression $[x{:}\Sigma_1]\mathtt{type}$ is not typable, just as `type` is not typable. In a translation to a PTS, these expressions should get type $\Box$;
3. There is a new reduction relation on expressions, which is specific for AUT-QE (which we call $\rightarrow_{\mathrm{QE}}$ in the sequel). This relation is given by the rule $[x_1{:}\Sigma_1]\cdots[x_n{:}\Sigma_n][y{:}\Omega]\mathtt{type} \rightarrow_{\mathrm{QE}} [x_1{:}\Sigma_1]\cdots[x_n{:}\Sigma_n]\mathtt{type}$ (for $n \geq 0$).

The first two rules are rather straightforward. They correspond to an extension of $\lambda{\rightarrow}$ to λP in Pure Type Systems. It is also easy to extend λ68 with similar rules: We just add the Π-formation rule $(*, \Box, \Box)$:

$$\frac{\Delta;\Gamma \vdash A : * \qquad \Delta;\Gamma, x{:}A \vdash B : \Box}{\Delta;\Gamma \vdash (\Pi x{:}A.B) : \Box}.$$

In AUT-68, PAT is implemented in de Bruijn-style (see Section 2.1 and Example 13). An implementation of predicate logic in Howard-style is not possible in AUT-68, but due to the extension with types of the form $[x{:}\Sigma]\mathtt{type}$, such an implementation becomes possible in AUT-QE. See [18].

The third rule deserves attention, as it is very unusual. It is needed in AUT-QE because that system does not distinguish λs and Πs. In AUT-68 this did not matter, as from the context it could always be derived whether an expression

$[x{:}\Sigma]\Omega$ should be interpreted as $\lambda x{:}\Sigma.\Omega$ or as $\Pi x{:}\Sigma.\Omega$. The latter should have type `type`, and the first should not have type `type`. In AUT-QE the situation is more complicated. An expression $[x{:}\Sigma]\Omega$ may have more than one type:

Example 56 Let $\mathfrak{B}$ consist of two lines:

$$(\varnothing, \alpha, —, \texttt{type}),$$
$$(\alpha{:}\texttt{type}, x, —, \alpha).$$

Notice that, using rule (abstr.1) of Definition 15, we can derive that

$$\mathfrak{B}; \alpha{:}\texttt{type} \vdash_{\mathrm{QE}} [x{:}\alpha]\alpha : \texttt{type}. \tag{21}$$

But using the new abstraction rule of AUT-QE we can also derive

$$\mathfrak{B}; \alpha{:}\texttt{type} \vdash_{\mathrm{QE}} [x{:}\alpha]\alpha : [x{:}\alpha]\texttt{type}. \tag{22}$$

More generally, we can prove that the two statements below are equivalent in AUT-QE (that is: if either of them is derivable then they are both derivable):

$$\mathfrak{B}; \Gamma \quad \vdash_{\mathrm{QE}} \quad [x_1{:}\Sigma_1]\cdots[x_n{:}\Sigma_n]\Omega : [x_1{:}\Sigma_1]\cdots[x_n{:}\Sigma_n]\texttt{type}; \tag{23}$$
$$\mathfrak{B}; \Gamma \quad \vdash_{\mathrm{QE}} \quad [x_1{:}\Sigma_1]\cdots[x_n{:}\Sigma_n]\Omega : [x_1{:}\Sigma_1]\cdots[x_m{:}\Sigma_m]\texttt{type} \tag{24}$$

(for $m < n$). In (23), the expression $[x_1{:}\Sigma_1]\cdots[x_n{:}\Sigma_n]\Omega$ should be read as $\lambda_{i=1}^{n}\, x_i{:}\Sigma_i.\Omega$; in (24) it should be read as $\lambda_{i=1}^{m}\, x_i{:}\Sigma_i.\prod_{j=m+1}^{n} x_j{:}\Sigma_j.\Omega$.

But this equivalence holds only for expressions of the form

$$[x_1{:}\Sigma_1]\cdots[x_n{:}\Sigma_n]\Omega$$

and not for general expressions Σ (take, for instance, Σ a variable). In order that the equivalence holds for general expressions Σ, de Bruijn introduced a rule for type inclusion:

$$\frac{\mathfrak{B}; \Gamma \vdash_{\mathrm{QE}} \Sigma : [x_1{:}\Sigma_1]\cdots[x_n{:}\Sigma_n]\texttt{type}}{\mathfrak{B}; \Gamma \vdash_{\mathrm{QE}} \Sigma : [x_1{:}\Sigma_1]\cdots[x_{n-1}{:}\Sigma_{n-1}]\texttt{type}}.$$

Lists of abstractions $[x_1{:}\Sigma_1]\cdots[x_n{:}\Sigma_n]$ were also called *telescopes* by de Bruijn. In the rule for type inclusion, we see that one part of the telescope "collapses".

5.3 $\Delta\Lambda$

As we saw above, de Bruijn departed from the classical notation of the λ-calculus and wrote the argument before the function and used $[x : A]$ instead of $\lambda x : A$ or $\Pi x : A$. So for example, de Bruijn wrote $\langle z\rangle[x : *][y : x]y$ instead of $(\lambda x : *.\lambda y : x.y)z$.

De Bruijn called items of the form $\langle B\rangle$ and $[x : C]$, A- (for application) respectively T- (for typing) *wagons*. De Bruijn called $\langle B\rangle[x : C]$, an AT-pair.

In de Bruijn's notation, the β-rule $(\lambda x : C.A)B \to_\beta A[x := B]$ becomes:

$$\langle B\rangle[x : C]A \to_\beta [x := B]A$$

Note that the A-wagon $\langle B\rangle$ and the T-wagon $[x:C]$ occur NEXT to each other.

Here is an example which compares β-reduction in both the classical and the de Bruijn notation. Wagons that have the same symbol on top, are matched (we ignore types for the sake of simplicity):

Classical Notation	De Bruijn's Notation
$\underline{(\overset{\circ}{\lambda}x\,.(\overset{+}{\lambda}y\,.\,\overset{-}{\lambda}z\,.zD)\,\overset{+}{C})\,\overset{\circ}{B})}\,\overset{-}{A}$	$\overset{-}{\langle A\rangle}\,\underline{\overset{\circ}{\langle B\rangle}[x]}\,\overset{\circ}{\langle C\rangle}\overset{+}{[y]}\overset{+}{[z]}\,\overset{-}{\langle D\rangle}z$
$\downarrow_\beta$	$\downarrow_\beta$
$\underline{((\overset{+}{\lambda}y\,.\,\overset{-}{\lambda}z\,.zD)\,\overset{+}{C})}\,\overset{-}{A}$	$\overset{-}{\langle A\rangle}\,\underline{\overset{+}{\langle C\rangle}[y]}\,\overset{-}{[z]}\,\overset{-}{\langle D\rangle}z$
$\downarrow_\beta$	$\downarrow_\beta$
$\underline{(\overset{-}{\lambda}z\,.zD)\,\overset{-}{A}}$	$\underline{\overset{-}{\langle A\rangle}[z]}\overset{-}{\langle D\rangle}z$
$\downarrow_\beta$	$\downarrow_\beta$
AD	$\langle D\rangle A$

The *bracketing structure* in classical notation of $((\overset{\circ}{\lambda}x\,.(\overset{+}{\lambda}y\,.\,\overset{-}{\lambda}z\,.zD)\,\overset{+}{C})\,\overset{\circ}{B})\,\overset{-}{A})$, is $\overset{\circ}{[}_1\,\overset{+}{[}_2\,\overset{-}{[}_3\,\overset{+}{]}_2\,\overset{\circ}{]}_1\,\overset{-}{]}_3$, where $[_i$ and $]_i$ match. Whereas $\overset{-}{\langle A\rangle}\overset{\circ}{\langle B\rangle}[x]\overset{\circ}{\langle C\rangle}\overset{+}{[y]}\overset{+}{[z]}\,\overset{-}{\langle D\rangle}z$ has the simpler bracketing structure $\overset{-\circ}{[}\,[\,\overset{\circ+}{]}\,\overset{+-}{[}\,]\,]$ or even better: $[\,[\,][\,]\,]$ in de Bruijn's notation. An A-wagon $\langle B\rangle$ and a T-wagon $[x:C]$ are *partners* when they match. Non-partnered wagons are *bachelors*. A sequence of wagons is called a *segment*. A segment is *well balanced* when it contains only partnered wagons.

Moreover, de Bruijn defined local β-reduction, which keeps the AT-pair and does β-reduction at one instance (instead of all the instances). For example (we take a simpler example than above and again ignore types for simplicity):

$\langle y\rangle[x]\langle x\rangle x$ β-reduces locally to $\langle y\rangle[x]\langle x\rangle y$ and to $\langle y\rangle[x]\langle y\rangle x$. Doing a further local β-reduction gives $\langle y\rangle[x]\langle y\rangle y$. Now that the $[x]$ does not bind any variable any more, and hence we can remove the AT-pair $\langle y\rangle[x]$ obtaining $\langle y\rangle y$.

Furthermore, de Bruijn generalised the AT-pair to the AT-couple where for example, in $\langle A\rangle\langle B\rangle[x]\langle C\rangle[y][z]\langle D\rangle z$, we have the AT-pairs: $\langle B\rangle[x]$ and $\langle C\rangle[y]$ and the AT-couple $\langle A\rangle[z]$. This definition of AT-couples leads to a natural generalisation of β-reduction as follows:

$\langle B\rangle\overline{s}[x:C]A \rightsquigarrow_\beta \overline{s}[x:=B]A$ where $\overline{s}$ is a well balanced segment.

So for example, $\langle A\rangle\langle B\rangle[x]\langle C\rangle[y][z]\langle D\rangle z \rightsquigarrow_\beta \langle B\rangle[x]\langle C\rangle[y][z:=A]\langle D\rangle z$.

The λ-calculus à la de Bruijn has many advantages over the classical λ-calculus. Some of these advantages are summarised in [37].

In AUT-SL (cf. B.2 of [44]), de Bruijn described how a complete AUTOMATH book can be written as a single lambda calculus formula. The disadvantage of AUT-SL was that in order to put the book into the lambda calculus framework, it was necessary to first eliminate all definitional lines of the book. De Bruijn did not like this idea as without definitions, formulae can exponentially grow.

For this reason, de Bruijn developped the $\Delta\Lambda$ calculus (cf. B.7 of [44]), with which he attempts to embrace all essential aspects of AUTOMATH apart from

type inclusion. $\Delta\Lambda$ is the lambda calculus written in his notation (as above)[15] but where β-reduction[16] is presented as the result of local β-reductions and AT-removals. The reason for this is that the delta reductions of AUTOMATH can be considered as local β-reductions, and not as ordinary β-reductions.

We have fully investigated PTSs and the type free lambda calculus in de Bruijn's notation [35, 37, 7]. We have also shown that $\rightsquigarrow_\beta$ satisfies nice properties in the type free lambda calculus [29] and that it loses subject reduction in PTSs but that subject reduction can be regained if definitions are added in the contexts [7]. We have not yet studied PTSs with local β-reductions and AT-removal, although we have studied the type free lambda calculus with local β-reduction, AT-removal and explicit substitution [34]. We leave the study of PTSs with de Bruijn's local β-reduction and AT-removal for future work.

6 Conclusion

In this paper we described the most basic AUTOMATH-system, AUT-68, in a PTS style. Though an attempt at such a description has been given before in [2, 22], we feel that our description is more accurate. Moreover, unlike [2, 22], our description pays attention to the definition and parameter systems, which are crucial in AUTOMATH. We gave a PTS called $\lambda 68$ which is closely related to AUT-68. Although $\lambda 68$ does not include Π-conversion (while AUTOMATH does), one can adapt it to include Π-conversion following the lines of [31].

The adaptation of $\lambda 68$ to a system λQE, representing the AUTOMATH-system AUT-QE is not hard, either: it requires adaptation of the Π-formation rule to include not only the rule $(*,*,*)$ but also $(*,\Box,\Box)$ and the introduction of the additional reduction rule of type inclusion. We leave this as a future work. We also leave as a future work the extension of PTSs with local β-reduction and AT-removal à la de Bruijn and hence the connection between de Bruijn's $\Delta\Lambda$ and PTSs with definitions.

There is no doubt that AUTOMATH has had an amazing influence in theorem proving, type theory and logical frameworks. AUTOMATH however, was developed independently from other developments in type theory and uses a λ-calculus and type-theoretical style that is unique to AUTOMATH. Writing AUTOMATH in the modern style of type theory will enable useful comparisons between type systems to take place. There are still many lessons to learn from AUTOMATH and writing it in modern style is a useful step in this direction.

References

[1] H.P. Barendregt. *The Lambda Calculus: its Syntax and Semantics*. Studies in Logic and the Foundations of Mathematics 103. North-Holland, Amsterdam, revised edition, 1984.

[2] H.P. Barendregt. λ-calculi with types. In *Handbook of Logic in Computer Science*, pages 117–309. OUP, 1992.

[15] In $\Delta\Lambda$, de Bruijn favours trees over character strings and does not make use of AT-couples.
[16] Recall this is now both β- and Π-reduction as he unifies λs and Πs.

[3] L.S. van Benthem Jutting. *Checking Landau's "Grundlagen" in the Automath system.* PhD thesis, Eindhoven University of Technology, 1977. Published as Mathematical Centre Tracts nr. 83, (Amsterdam 1979).

[4] L.S. van Benthem Jutting. Description of AUT-68. Technical Report 12, Eindhoven University of Technology, 1981. Also in [44], pp. 251–273.

[5] S. Berardi. Towards a mathematical analysis of the Coquand-Huet calculus of constructions and the other systems in Barendregt's cube. Technical report, Dept. of Computer Science, Carnegie-Mellon University and Dipartimento Matematica, Universita di Torino, 1988.

[6] R. Bloo, F. Kamareddine, L. Laan, and R.P. Nederpelt. *Parameters in Pure Type Systems*, volume 2286 of *Lecture Notes in Computer Science*, pages 371–385. Springer Verlag, 2002.

[7] R. Bloo, F. Kamareddine, and R.P. Nederpelt. The Barendregt Cube with Definitions and Generalised Reduction. *Information and Computation*, 126(2):123–143, 1996.

[8] L.E.J. Brouwer. *Over de Grondslagen der Wiskunde.* PhD thesis, Universiteit van Amsterdam, 1907. Dutch; English translation in [27].

[9] N.G. de Bruijn. AUTOMATH, a language for mathematics. Technical Report 68-WSK-05, T.H.-Reports, Eindhoven University of Technology, 1968.

[10] N.G. de Bruijn. The mathematical language AUTOMATH, its usage and some of its extensions. In M. Laudet, D. Lacombe, and M. Schuetzenberger, editors, *Symposium on Automatic Demonstration*, pages 29–61, IRIA, Versailles, 1968. Springer Verlag, Berlin, 1970. Lecture Notes in Mathematics **125**; also in [44], pages 73–100.

[11] N.G. de Bruijn. The Mathematical Vernacular, a language for mathematics with typed sets. In P. Dybjer et al., editors, *Proceedings of the Workshop on Programming Languages.* Marstrand, Sweden, 1987. Reprinted in [44] in combination with *Formalizing the Mathematical Vernacular* (formerly unpublished, 1982).

[12] N.G. de Bruijn. Reflections on Automath. Eindhoven University of Technology, 1990. Also in [44], pages 201–228.

[13] A. Church. A set of postulates for the foundation of logic (1). *Annals of Mathematics*, 33:346–366, 1932.

[14] A. Church. A set of postulates for the foundation of logic (2). *Annals of Mathematics*, 34:839–864, 1933.

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

[16] R.L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, New Jersey, 1986.

[17] H.B. Curry and R. Feys. *Combinatory Logic I.* Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam, 1958.

[18] 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, pages 48–77, 1973. Also in [44], pages 101–126.

[19] D.T. van Daalen. *The Language Theory of Automath.* PhD thesis, Eindhoven University of Technology, 1980.

[20] G. Dowek et al. The Coq Proof Assistant Version 5.6, Users Guide. Technical Report 134, INRIA, Le Chesney, 1991.

[21] G. Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens.* Nebert, Halle, 1879. Also in [24], pages 1–82.

[22] J.H. Geuvers. *Logics and Type Systems.* PhD thesis, Catholic University of Nijmegen, 1993.

[23] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. In *Proceedings Second Symposium on Logic in Computer Science*, pages 194–204, Washington D.C., 1987. IEEE.

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

[25] A. Heyting. *Mathematische Grundlagenforschung. Intuitionismus. Beweistheorie*. Ergebnisse der Mathematik und ihrer Grenzgebiete. Springer Verlag, Berlin, 1934.

[26] A. Heyting. *Intuitionism, an introduction*. Studies in Logic and the Foundations of Mathematics. North Holland, Amsterdam, 1956.

[27] A. Heyting, editor. *Brouwer: Collected Works*, volume 1. North-Holland, Amsterdam, 1975.

[28] W.A. Howard. The formulas-as-types notion of construction. In [46], pages 479–490, 1980.

[29] F. Kamareddine. Postponement, conservation and preservation of strong normalisation for generalised reduction. *Journal of Logic and Computation*, 10(5):721–738, 2000.

[30] F. Kamareddine. *On Functions and Types: A Tutorial*, volume 2540 of *Lecture Notes in Computer Science*, pages 74–93. Springer Verlag, 2002.

[31] 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 Logics*, 97:27–45, 1999.

[32] F. Kamareddine, L. Laan, and R.P. Nederpelt. Refining the Barendregt cube using parameters. *Fifth International Symposium on Functional and Logic Programming, FLOPS 2001*, LNCS 2024:375–389, 2001.

[33] F. Kamareddine, L. Laan, and R.P. Nederpelt. Revisiting the notion of function. *Algebraic and Logic Programming*, 54:65–107, 2003.

[34] F. Kamareddine and R.P. Nederpelt. On stepwise explicit substitution. *International Journal of Foundations of Computer Science*, 4:197–240, 1993.

[35] F. Kamareddine and R.P. Nederpelt. Refining reduction in the λ-calculus. *Journal of Functional Programming*, 5(4):637–651, October 1995.

[36] F. Kamareddine and R.P. Nederpelt. Canonical typing and Π-conversion in the Barendregt Cube. *Journal of Functional Programming*, 6(2):245–267, 1996.

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

[38] S.C. Kleene and J.B. Rosser. The inconsistency of certain formal logics. *Annals of Mathematics*, 36:630–636, 1935.

[39] J.W. Klop. Term rewriting systems. In *Handbook of Logic in Computer Science*, pages 1–116. UP, 1992.

[40] A.N. Kolmogorov. Zur Deutung der Intuitionistischen Logik. *Mathematisches Zeitschrift*, 35:58–65, 1932.

[41] T. Laan. *The Evolution of Type Theory in Logic and Mathematics*. PhD thesis, Eindhoven University of Technology, 1997.

[42] E. Landau. *Grundlagen der Analysis*. Leipzig, 1930.

[43] R.P. Nederpelt. Presentation of natural deduction. *Recueil des travaux de l'Institut Mathématique, Nouvelle série*, 2(10):115–126, 1977. Symposium: Set Theory. Foundations of Mathematics, Beograd 1977.

[44] 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.

[45] M.J. O'Donnell. *Computing in Systems Described by Equations*, volume 58 of *Lecture Notes in Computer Science*. Springer Verlag, 1977.

[46] J.P. Seldin and J.R. Hindley, editors. *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism.* Academic Press, New York, 1980.

[47] P. Severi and E. Poll. Pure type systems with definitions. Technical Report 24, TUE Computing Science Notes, Eindhoven University of Technology, 1993.

[48] T. Streicher. *Semantics of Type Theory.* Birkhäuser, 1991.

[49] W.W. Tait. Infinitely long terms of transfinite type. In J.N. Crossley and M.A.E. Dummett, editors, *Formal Systems and Recursive Functions*, Amsterdam, 1965. North-Holland.

[50] J. Terlouw. Een nadere bewijstheoretische analyse van GSTT's. Technical report, Department of Computer Science, University of Nijmegen, 1989.

[51] A.N. Whitehead and B. Russell. *Principia Mathematica*, volume I, II, III. Cambridge University Press, 1910, 1912, 1913[1], 1925, 1925, 1927[2].

[52] J. Zucker. Formalization of classical mathematics in Automath. In *Colloque International de Logique,* Clermont-Ferrand, pages 135–145, Paris, CNRS, 1977. Colloques Internationaux du Centre National de la Recherche Scientifique, 249.