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@cs.ewi.nl
†Weerdseste 45, 3431 L S Nieuwegen, The Netherlands. Email: twan.laan@er.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 Bentham 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 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 interpretation of the logical connectives (see [26, 40]). The interpretation of the proof of an implication $A \rightarrow B$ 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 $\lambda$-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 $\lambda$-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

---

*The first practical use of the propositions-as-types principle is found in AUTOMATH.*
“Begriffsschrift” [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 influential “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 Bernardi [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 Πx: A.B and λx: 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: (Πx: A.B)N →Π B[x:=N]. We leave the features of Π-application and Π-reduction till Section 5.
- In Section 4, we give a system λ68 that is (almost) a PTS. In λ68, definitions play an active role. We show that λ68 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 λ68 either, but parameters are hidden in the rules for the construction of product types.
- In Section 5 we discuss how λ68 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 λx:A.B or Πx:A.B in PTSs. In addition to Aut-QE, we reflect on ∆A (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.

3
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 \( \land, \lor \)', 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, there are frequently occurring constructions in intuitionistic logic.
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 \textit{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 \textit{types} are used to represent these sets of proofs. An element of such a set of proofs is represented as a \textit{term} of the corresponding type. This means that propositions are interpreted as \textit{types}, and proofs of a proposition $a$ as \textit{terms of type $a$}.

\section*{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 $\varnothing_1, \varnothing_2, \ldots$ is chosen. All these primitive objects are F-objects. Moreover, if $\alpha$ and $\beta$ are F-objects, then so is $\alpha \beta$. Here, $F$ is a new symbol. $\alpha \beta$ must be interpreted as the class of functions from $\alpha$ to $\beta$. If $\alpha$ is an F-object, then the statement $\vdash \alpha X$ must be interpreted as “the object $X$ belongs to $\alpha$”. The rule-F is adopted: if $\vdash FXYZ$ and $\vdash XU$ then $\vdash Y(\mathcal{U})$. This rule immediately corresponds to the application-rule of Church's $\lambda$-calculus and says: if $Z$ belongs to $FXY$ and $U$ belongs to $X$, then $\mathcal{U}$ belongs to $Y$.

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

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

\begin{flushright}
\textit{Een logische opbouw der wiskunde, onafhankelijk van de wiskundige intuïtie, is onmogelijk — daar op die manier slechts een taalgebouw wordt verkeken, 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}.
\end{flushright}

\textit{(Over de Grondslagen der Wiskunde [8], p. 180)}

\begin{flushright}
\textit{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 contradiction in terms — because a logical system needs the basic intuition of mathematics as much as mathematics itself needs it. [Translation from [27]].}
\end{flushright}

\textsuperscript{3}Remark that Curry's function $\alpha \mapsto \alpha^\mathcal{P}$ 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 \( F_mX_1 \cdots X_nY \) is an abbreviation of \( F(X_1 (F_2 \cdots (F_{X_m}(Y) \cdots ))) \) then:

\[
\text{"If } \vdash \forall_m \xi_1 \cdots \xi_n \eta Y \text{ then } \vdash (F_m \xi_1 \cdots \xi_n \eta)^p. \text{ Moreover, if } \vdash \forall_m \xi_1 \cdots \xi_n \eta X \text{ is derivable from the premises } \vdash \alpha \eta_i (i = 1, \ldots, p) \text{ then } \vdash (F_m \xi_1 \cdots \xi_n \eta)^p \text{ is derivable from the premises } \vdash \alpha_i^p (i = 1, \ldots, p).\"
\]

([17], paragraph 9E, Theorem 1)

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

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 \( (F_m \xi_1 \cdots \xi_n \eta)^p \). But this is not made explicit in [17].

**Example 1** As an example, we show the deduction of the proposition \( A \rightarrow A \) from the logical axioms \( X \rightarrow Y \rightarrow X^4 \) (the K-axiom) and \( (X \rightarrow Y \rightarrow Z) \rightarrow (X \rightarrow Y) \rightarrow X \rightarrow Z \) (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 \( A \rightarrow A \) in natural deduction style, with the use of modus ponens, and axioms \( X \rightarrow Y \rightarrow X \) and \( (X \rightarrow Y \rightarrow Z) \rightarrow (X \rightarrow Y) \rightarrow X \rightarrow Z \) only:

\[
\begin{align*}
\vdash (A \rightarrow (A \rightarrow A) \rightarrow A) & \rightarrow (A \rightarrow A) \rightarrow A \rightarrow A \\
\vdash A \rightarrow (A \rightarrow A) \rightarrow A & \rightarrow (A \rightarrow A) \\
\vdash (A \rightarrow A) \rightarrow A & \rightarrow A \\
\vdash A \rightarrow A & \rightarrow A
\end{align*}
\]

- We use \( P_mX_1 \cdots X_nY \) as an abbreviation for \( P_1X_1 (P_2X_2 (\cdots (P_mX_mY) \cdots )) \). So \( P_mX_1 \cdots X_nY \) can be interpreted as the proposition \( X_1 \rightarrow X_2 \rightarrow \cdots \rightarrow X_m \rightarrow Y \). In this notation, Rule-P is:

\[
\vdash P_{m+1}X_0 \cdots X_mY \rightarrow P_mX_1 \cdots X_mY 
\]

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

- **K** (K): \( \vdash P_2XY \cdot X \).
- **S** (S): \( \vdash (P_2XYZ)(PXY) \cdot XZ \).

Let \( A \) be a term. From the axioms we derive \( \vdash PA \), using rule-P:

\[
\begin{align*}
\vdash P_3(P_2A(P(AA))A)(P(A(PAA))A) & \rightarrow P_2A(P(AA))A \\
\vdash P_2(P(A(PAA))AA) & \rightarrow PA(PAA) \\
\vdash PAA & \rightarrow PA(PAA)
\end{align*}
\]

(\( \text{so a types-as-propositions embedding instead of a propositions-as-types embedding.} \))
• 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 $\lambda$-term $\lambda x. y. x$) 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 $(F_2XYX)^P$. This is indicated by putting $K$ behind the axiom: $(F_2XYX)^P K$. The combinator $S$, comparable to the $\lambda$-term $\lambda xyz. xz(yz)$, has type $F_3(F_2XYZ)(FXY)XZ$ for arbitrary $F$-objects $X, Y, Z$. $S$ is a “proof” of the axiom $(F_3(F_2XYZ)(FXY)XZ)^P$. This is denoted as $(F_3(F_2XYZ)(FXY)XZ)^PS$. The derivation above now translates to:

$$
\vdash F_3(F_2A(FAA)A)((F(A(FAA)))AAPK)
\vdash F_2A(FAA)AK
\vdash F_2(A(FAA))A(A(SK)) \vdash F(AA)K
\vdash FAA(SKK)
$$

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 $A \rightarrow A$.

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

$$
\vdash (A \rightarrow (A \rightarrow A) \rightarrow A) \rightarrow (A \rightarrow A \rightarrow A) \rightarrow A \rightarrow A
\vdash A \rightarrow (A \rightarrow A) \rightarrow A
\vdash (A \rightarrow A \rightarrow A) \rightarrow A \rightarrow A
\vdash A \rightarrow A \rightarrow A
\vdash A \rightarrow A
$$

**Howard**

Howard [28] combines the argument of Curry and Feys [17] with Tait’s discovery of the correspondence between cut elimination and $\beta$-reduction of $\lambda$-terms [49].

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

$$
\frac{[A]}{D_1} \quad \frac{B}{A \rightarrow B} \quad \frac{D_2}{A} \quad \frac{[A]}{B}
$$

Here, $[A]$ denotes that the assumption $A$ has been discharged at the point where we concluded $A \rightarrow B$ from $B$. $D_1$ is a derivation with some assumptions of $A$, and conclusion $B$, whilst $D_2$ is a derivation with conclusion $A$. The derivation $D_2$ can be used to replace the assumptions of $A$ in derivation $D_1$. This means that we can transform the derivation to:
\[
\begin{array}{c}
\Omega_2 \\
\alpha \\
\Omega_1 \\
\beta
\end{array}
\]

where copies of \( \Omega_2 \) have replaced the assumptions \( A \) in \( \Omega_1 \).

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

\[
\frac{[x:A]}{T : B} \quad \frac{\Omega_1}{\lambda x : A . T : (A \rightarrow B) \quad \Omega_2} \quad \frac{S : A}{((\lambda x : A . T) \cdot S) : B}
\]

and

\[
\frac{\Omega_2}{S : A} \quad \frac{T [x := S] : B}{\Omega_1}
\]

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 \( \Omega_1 \) is represented by a \( \lambda \)-term \( T \), in which the variable \( x \) may occur (we can use the assumption \( A \) in derivation \( \Omega_1 \)). Then the term \( \lambda x : A . T \) exactly represents a proof of \( A \rightarrow B \); it is a function that transforms any proof \( x \) of \( A \) into a proof \( T \) of \( B \). As \( \Omega_2 \) 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 \( (\lambda x : A . T) \cdot S \) of \( B \).

Substituting the derivation \( \Omega_2 \) for the assumptions of \( A \) in \( \Omega_1 \) 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 \( \lambda \)-term \( T [x := S] \). The proof transformation exactly corresponds to the \( \beta \)-reduction \( (\lambda x : A . T) \cdot S \rightarrow_\beta T [x := S] \).

This is the first time that proofs are treated as \( \lambda \)-terms. Howard doesn’t call these \( \lambda \)-terms “proofs” but “constructions”. Moreover, Howard’s treatment of \( \text{PAT} \) pays attention to both Propositions as Types (following the line of Curry and Feys) and Proofs as Terms (by using \( \lambda \)-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 \( \text{PAT} \) in the first \textsc{Automath} system of de Bruijn (\textsc{AUT-68} \cite{44}, \cite{10}). Though de Bruijn was probably influenced by Heyting (see \cite{12} in \cite{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 \( \text{bool} \) is introduced. \( \text{bool} \) is a type: the type of propositions. If \( b \) is a term of type \( \text{bool} \) (so \( b \) is a proposition), then \( \text{true}(b) \) is a primitive notion of type type. \( \text{true}(b) \) represents the type of the proofs of \( b \). So, a proof of proposition \( b \) is of type \( \text{true}(b) \) 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 \( \text{PAT} \). 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 \( \text{AUTOMATH} \) systems such as \( \text{AUT-68} \) one usually finds a “bool-style” implementation of \( \text{PAT} \). It would be impossible to give a “prop-style” implementation in such a system as its \( \lambda \)-calculus is not strong enough to support it. In \( \text{AUTOMATH} \) systems with a more powerful \( \lambda \)-calculus we also find “prop-style” implementations. See [43] for a description of prop-style implementations in \( \text{AUTOMATH} \).

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 \( \text{PAT} \) in a bool-style way (see [12]).

Though the bool-style implementation is not used in later \( \text{AUTOMATH} \) systems, it 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 PTVs. In this section, we give the necessary machinery of PTVs 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 \( T(\mathbb{V}, \mathbb{C}) \) (or \( T \), 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: $T ::= \mathbb{V} | \mathbb{C} | TT | \lambda V.T.T | \Pi V.T.T.$

We use $x, y, z, \alpha, \beta$ 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: $x$ is a specific element of $\mathbb{V}$; while $x$ is a meta-variable over $\mathbb{V}$. The variables $x, y, z$ are assumed to be distinct elements of $\mathbb{V}$ (so $x \neq y$ 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 $T$. $\text{fv}(A)$, 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 $\pi \xi:A.B$, or $\pi_{i=1}^n x_i:A_i.A$.
  - We write $A[x:=B]$ as $A[x:=B]_{i=m}^n$. If $m > n$ then $A[x_i:=B_i]_{i=m}^n$ denotes $A[x_i:=B_i]_{i=1}^n$ as $A[x:=B]$. We also write $A[x_i:=B_i]_{i=m}^n$.

**Definition 5** *(\beta-reduction)* The relation $\rightarrow_\beta$ is given by the contraction rule $(\lambda x:A_1.A_2)B \rightarrow_\beta \Lambda_2[x:=B]$ and the usual compatibility rules. The relation $\rightarrow_\beta$ (resp. $\equiv_\beta$) is the smallest reflexive and transitive (resp. equivalence) relation that includes $\rightarrow_\beta$. By $A \rightarrow_\beta^+ B$ we indicate that $A \rightarrow_\beta B$, but $A \neq B$.

A term with no subterms of the form $(\lambda x:A_1.A_2)B$ in a $\beta$-normal form, or a normal form if no confusion arises. We write $A \rightarrow_\beta^N B$ (resp. $A \equiv_\beta^N B$) if $A \rightarrow_\beta B$ (resp. $A \equiv_\beta B$) and $B$ is in $\beta$-normal form.

**Definition 6** 
- A specification is a triple $(S, A, R)$, such that $S \subseteq \mathbb{C}$, $A \subseteq S \times S$ and $R \subseteq S \times S \times S$. The specification is single sorted if $A$ and $R$ are (partial) functions from $S \rightarrow S$ and $S \times S \rightarrow S$ resp. We call $S$ the set of sorts, $A$ the set of axioms, and $R$ the set of (I-formation) rules.

- A context is a finite (possibly empty) list $x_1:A_1, \ldots, x_n:A_n$ (or $\xi:A$) of variable declarations. $\{x_1, \ldots, x_n\}$ is the domain $\text{dom} \{\xi:A\}$ of the context. The empty context is denoted $\epsilon$. We use $\Gamma$, $\Delta$ to range over contexts.

- We extend substitutions to contexts by: $\epsilon[x:=A] \equiv \epsilon$; and $(\Gamma, y:B)[x:=A] \equiv \left\{ \begin{array}{ll} \Gamma[x:=A] & \text{if } x \equiv y; \\
\Gamma[x:=A], y:B[x:=A] & \text{if } x \neq y. \end{array} \right.$

Though PTs 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 5.3.4. of [18], and Section 2).
(axiom) \[
\emptyset \vdash s_1 : s_2 \quad (s_1, s_2) \in A
\]

(start) \[
\Gamma \vdash A : s \quad x \notin \text{DOM} (\Gamma)
\]

(weak) \[
\Gamma, x : A \vdash x : A \\
\Gamma \vdash A : B \quad \Gamma \vdash C : s \\
\Gamma, x : C \vdash A : B \\
x \notin \text{DOM} (\Gamma)
\]

(\Pi) \[
\Gamma \vdash A : s_1 \\
\Gamma \vdash (\Pi x : A.B) : s_2 \\
\Gamma, x : A \vdash b : B \\
\Gamma \vdash (\Pi x : A.B) : s
\]

(\lambda) \[
\Gamma \vdash \lambda x : A.b : (\Pi x : A.B) \\
\Gamma \vdash a : A \\
\Gamma \vdash Fa : B[x:=a]
\]

(appl) \[
\Gamma \vdash A : B \\
\Gamma \vdash B' : s \\
\Gamma \vdash A : B'
\]

\[
B = \beta B'
\]

Figure 1: Typing rules of PTs

| λA → | [s, s] |
| λA2 | [s, s] [□, s] |
| λAP | [s, s] [(s, □)] |
| λAω | [s, s] (□, □) |
| λAω2 | [s, s] [□, s] [(s, □)] |
| λAω3 | [s, s] (□, □) |
| λC | [s, s] [□, s] [□, □] |

Figure 2: The Barendregt Cube

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

An important class of PTs is given as eight PTs in the Barendregt Cube [2] of Figure 2. These systems all have \( \{s, □\} \) as set of sorts, and \( s : □ \) as only axiom, but differ on the \( \Pi \)-formation rules. We write \( (s_1, s_2, s_3) \) 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, $\mathcal{V} \cap \mathcal{C} = \emptyset$ and type $\not\in \mathcal{V} \cup \mathcal{C}$. 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 $\mathcal{E}^+ \overset{\text{def}}{=} \mathcal{E} \cup \{\text{type}\}$.

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

(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 $x \in \mathcal{V}$, $\Sigma \in \mathcal{E} \cup \{\text{type}\}$ 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 $\lambda$-calculus, like AUT-QE.

- The AUT-68-expression $\langle \Sigma_2 \rangle \Sigma_1$ is AUTOMATH-notation for the intended application of the “function” $\Sigma_1$ to the “argument” $\Sigma_2$. In PTS-notation: $\Sigma_1 \Sigma_2$. (Note the unusual order of “function” $\Sigma_1$ and “argument” $\Sigma_2$). 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 $\Sigma_1$ 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 $\Sigma_2$ and the abstraction $[x: \Omega_1]$ belong together: as soon as the intended application of the function $\Sigma_1$ to its argument is carried out, $\Sigma_2$ is substituted for $x$ everywhere in $\Omega_2$. It is convenient to put expressions that belong together next to each other. In classical notation, one writes $(x: \Omega_1)[x: \Omega_2]$, where $\Sigma_2$ and $[x: \Omega_1]$ are separated from each other by the expression $\Omega_2$. This makes the structure of the expression less clear, in particular if $\Omega_2$ is a very long expression.

We define $\text{fv}(A)$ in the same way as for PTSs where also $\text{fv}(a(\Sigma_1, \ldots, \Sigma_n)) \overset{\text{def}}{=} \bigcup_{i=1}^n \text{fv}(\Sigma_i)$. 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 $\Omega$ 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 $\Omega$). Moreover, $\text{type}[x_1, \ldots, x_n:=\Sigma_1, \ldots, \Sigma_n]$ is defined as 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,
• $\Gamma$ is a context, i.e. a finite (possibly empty) list $x_1;\alpha_1, \ldots, x_n;\alpha_n$, where the $x_i$s are different elements of $V$ and the $\alpha_i$s are elements of $E \cup \{\text{type}\}$;

• $\Sigma_1$ can be (only): o The symbol $\alpha$ (if $k \in V$); o The symbol $PN$ (if $k \in C$) ($PN$ stands for “primitive notion”); o An element of $E$ (if $k \in C$);

• $k$ is an element of $V \cup C$; and $\Sigma_2$ is an element of $E \cup \{\text{type}\}$.

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

1. $(\Gamma; k; \ldots; \Sigma_2)$ with $k \in V$. This is a variable declaration of the variable $k$ having type $\Sigma_2$. 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. $(\Gamma; k; PN; \Sigma_2)$ with $k \in C$. This line introduces a primitive notion: a constant $k$ of type $\Sigma_2$. 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 $\Gamma$. 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 $\text{and}(A, B)$

5 Instead, we want to have one primitive notion $\text{and}$, to which we can add two propositions $A$ and $B$ as parameters when we want to form the proposition $\text{and}(A, B)$. Therefore, we introduce $\text{and}$ in a context $\Gamma \equiv x: \text{prop}, y: \text{prop}$. Given certain propositions $A, B$ this enables us to form the Aut-68-expression $\text{and}(A, B)$;

3. $(\Gamma; k; \Sigma_1; \Sigma_2)$ with $k \in C$ and $\Sigma_1 \in E$. This line introduces a definition. The definiendum $k$ is defined by the definiens $\Sigma_1$ and has type $\Sigma_2$. 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 $S(S(S(S(S(S(0)))))))$ 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:

\footnote{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 $x$ and $y$ represents only one variable, and not the application of $a$ to $x$ and $a$.}
<table>
<thead>
<tr>
<th></th>
<th>prop</th>
<th>PN</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊤</td>
<td>prop</td>
<td>PN</td>
<td>prop</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
<td>⊥</td>
<td>prop</td>
</tr>
<tr>
<td>x, y</td>
<td>and</td>
<td>⊥</td>
<td>prop</td>
</tr>
<tr>
<td>x</td>
<td>proof</td>
<td>⊥</td>
<td>type</td>
</tr>
<tr>
<td>x, y</td>
<td>pr</td>
<td>⊥</td>
<td>proof(x)</td>
</tr>
<tr>
<td>x, y, px</td>
<td>py</td>
<td>⊥</td>
<td>proof(y)</td>
</tr>
<tr>
<td>x, y, px, py</td>
<td>and-I</td>
<td>PN</td>
<td>proof(x and y)</td>
</tr>
<tr>
<td>x, y, px</td>
<td>and-01</td>
<td>PN</td>
<td>proof(x)</td>
</tr>
<tr>
<td>x</td>
<td>prx</td>
<td>⊥</td>
<td>proof(x)</td>
</tr>
<tr>
<td>x, prx</td>
<td>and-R</td>
<td>and-I(x, x, prx)</td>
<td>proof(and(x, x))</td>
</tr>
<tr>
<td>x, y, px</td>
<td>and-S</td>
<td>and-I(y, x, and-02, and-01)</td>
<td>proof(and(y, x))</td>
</tr>
</tbody>
</table>

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 ∈ 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, \( \text{proof}(\text{and}) \) in line 8 should be read as \( \text{proof}(\text{and}(x,y)) \).

Similarly, we can write \( \text{proof} \) instead of \( \text{proof}(x) \) in line 6. From line 5 (where \( \text{proof} \) is introduced) we find that \( \text{proof} \) carries one parameter. Writing just \( \text{proof} \) 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 \( \text{proof}(y) \) in line 7 because \( \text{proof} \) would give \( \text{proof}(x) \);

9. To express how we can use a proof of \( \text{and}(x,y) \), first we introduce a variable \( p_x \) that represents an arbitrary proof of \( \text{and}(x,y) \);

10. As we want \( x \) to hold when \( \text{and}(x,y) \) holds, we introduce an axiom \( \text{and}-01 \) (and-out, first and-elimination). Given propositions \( x,y \) and a proof \( p_y \) of \( \text{and}(x,y) \), \( \text{and}-01(x,y,p_y) \) is a proof of \( x \);

11. Similarly, we introduce an axiom \( \text{and}-02 \) representing a proof of \( y \);

- We can now derive some elementary theorems:

12. We want to derive \( \text{and}(x,x) \) from \( x \). That is: from a proof of \( x \), we can construct a proof of \( \text{and}(x,x) \). In line 6, we introduced a variable \( p_x \) for a proof of \( x \). However, we declared \( p_x \) 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 \( p_{rx} \), in the context \( x \);

13. We derive our theorem: the reflexivity of the logical conjunction. Given a proposition \( x \), and a proof \( p_{rx} \) of \( x \), we can use the axiom \( \text{and}-I \) to find a proof of \( \text{and}(x,x) \): we can use \( \text{and}-I(x,x,p_x,p_x) \) thanks to line 8. We give a name to this proof: \( \text{and}-R \). If, anywhere in the sequel of the book, \( \Sigma \) is a proposition, and \( \Omega \) is a proof of \( \Sigma \), we can write \( \text{and}-R(\Sigma, \Omega) \) for a proof of \( \text{and}(\Sigma, \Sigma) \). This is shorter, and more expressive, than the original expression \( \text{and}-I(\Sigma, \Sigma, \Omega, \Omega) \);

14. We also show that \( \text{and} \) is symmetric: whenever \( \text{and}(x,y) \) holds, we also have \( \text{and}(y,x) \). The idea is as follows. Given propositions \( x,y \) and a proof \( p_y \) of \( \text{and}(x,y) \), we can form proofs \( \text{and}-01(x,y,p_y,p_y) \) of \( x \) and \( \text{and}-02(x,y,p_y,p_y) \) of \( y \). We can read these proofs “in reverse order” to the axiom \( \text{and}-I \): the expression \( \text{and}-I(y,x,\text{and}-02,\text{and}-01) \) represents a proof of \( \text{and}(y,x) \). The expression \( \text{and}-02 \) should be read as \( \text{and}-02(x,y,p_y) \) due to the “default parameter” mechanism. Similarly, \( \text{and}-01 \) must be read as \( \text{and}-01(x,y,p_y) \).

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 \( \Sigma_1 \) and \( \Sigma_2 \) (as long as \( \Sigma_1 \) is not \( \mathfrak{P} \mathfrak{N} \) or \( \mathfrak{M} \) and \( \Sigma_2 \) is not \( \mathfrak{T} \mathfrak{y} \mathfrak{p} \mathfrak{e} \)) must be well-defined, i.e. the elements of \( \forall \cup \mathcal{C} \) occurring in them must have been established (as variables, primitive notions, or defined constants) in previous
parts of $\mathcal{B}$. The same holds for the type assignments $x_i: \alpha_i$ that occur in $\Gamma$. Moreover, if $\Sigma_1$ is not $\text{PN}$ or $-$, then $\Sigma_1$ must be of the same type as $k$, hence $\Sigma_1$ must be of type $\Sigma_2$ (within the context $\Gamma$). 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 $\mathcal{B}; \emptyset \vdash \text{ok}$ to indicate that a book $\mathcal{B}$ is correct, and $\mathcal{B}; \Gamma \vdash \text{ok}$ to indicate that the context $\Gamma$ is correct with respect to the (correct) book $\mathcal{B}$.

We write $\mathcal{B}; \Gamma \vdash \Sigma_1 : \Sigma_2$ to indicate that $\Sigma_1$ is a correct expression of type $\Sigma_2$ (or simply a correct expression) with respect to $\mathcal{B}$ and $\Gamma$. We also say: $\Sigma_1 : \Sigma_2$ is a correct statement with respect to $\mathcal{B}$ and $\Gamma$. 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 $\mathcal{B}$ and a context $\Gamma$ are correct if $\mathcal{B}; \Gamma \vdash \text{ok}$ can be derived with the rules below (= is given in Section 2.5. The rules use correct statements of Definition 15):

- **(axiom)**
  \[ \mathcal{B}; \emptyset \vdash \text{ok} \]

- **(context ext.)**
  \[ \mathcal{B}_1, (\Gamma; x; \text{--}; \alpha); \mathcal{B}_2; \Gamma \vdash \text{ok} \]
  \[ \mathcal{B}_1, (\Gamma; x; \text{--}; \alpha); \mathcal{B}_2; \Gamma, x: \alpha \vdash \text{ok} \]
  \[ \mathcal{B}_3; \Gamma \vdash \text{ok} \]

- **(book ext.: var1)**
  \[ \mathcal{B}, (\Gamma; x; \text{--}; \text{type}); \emptyset \vdash \text{ok} \]
  \[ \mathcal{B}, (\Gamma; x; \text{--}; \text{type}); \mathcal{B}_3; \Gamma \vdash \text{ok} \]

- **(book ext.: var2)**
  \[ \mathcal{B}, (\Gamma; x; \text{--}; \Sigma_2); \emptyset \vdash \text{ok} \]
  \[ \mathcal{B}_3; \Gamma \vdash \text{ok} \]

- **(book ext.: pn1)**
  \[ \mathcal{B}, (\Gamma; k; \text{PN}; \text{type}); \emptyset \vdash \text{ok} \]
  \[ \mathcal{B}_3; \Gamma \vdash \Sigma_2 : \text{type} \]

- **(book ext.: pn2)**
  \[ \mathcal{B}, (\Gamma; k; \text{PN}; \Sigma_2); \emptyset \vdash \text{ok} \]
  \[ \mathcal{B}_3; \Gamma \vdash \Sigma_2 : \text{type} \]

- **(book ext.: def1)**
  \[ \mathcal{B}, (\Gamma; k; \Sigma_1; \text{type}); \emptyset \vdash \text{ok} \]
  \[ \mathcal{B}_3; \Gamma \vdash \Sigma_1 : \text{type} \]

- **(book ext.: def2)**
  \[ \mathcal{B}; \Gamma \vdash \Sigma_2 : \text{type} \]
  \[ \mathcal{B}_3; \Gamma \vdash \Sigma_1 : \Sigma_2' \]
  \[ \mathcal{B}_3; \Gamma \vdash \Sigma_2 =_{\text{bd}} \Sigma_2' \]
  \[ \mathcal{B}_3; (\Gamma; k; \Sigma_1; \Sigma_2); \emptyset \vdash \text{ok} \]

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

**Definition 15 (Correct Statements)** A statement $\mathcal{B}; \Gamma : \Omega$ 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).

---

*As the empty context will be correct with respect to any correct book.*
(start) \[
\frac{\mathcal{B}; \Gamma_1, x \alpha, \Gamma_2 \vdash \text{ok}}{
\mathcal{B}; \Gamma_1, x \alpha, \Gamma_2 \vdash x \alpha}
\]

\(\mathcal{B} \equiv \mathcal{B}_1, (x_1: \alpha_1, \ldots, x_n: \alpha_n; b; \Omega_1; \Omega_2), \mathcal{B}_2\)

(parameters) \[
\frac{\mathcal{B}; \Gamma \vdash \Sigma; \alpha_i[x_1, \ldots, x_{i-1} := \Sigma_1, \ldots, \Sigma_{i-1}]/(i = 1, \ldots, n)}{
\mathcal{B}; \Gamma \vdash \Sigma; \alpha_i[x_1, \ldots, x_n := \Sigma_1, \ldots, \Sigma_n]}
\]

(abstr.1) \[
\frac{\mathcal{B}; \Gamma \vdash \Sigma; \text{type} \quad \mathcal{B}; \Gamma, x: \Sigma_1 \vdash \Omega_1; \text{type}}{
\mathcal{B}; \Gamma \vdash [x: \Sigma_1] \Omega_1 : \text{type}}
\]

(abstr.2) \[
\frac{\mathcal{B}; \Gamma \vdash \Sigma; \text{type} \quad \mathcal{B}; \Gamma, x: \Sigma_1 \vdash \Omega_1; \text{type} \quad \mathcal{B}; \Gamma, x: \Sigma_1 \vdash \Sigma_2; \Omega_1}{
\mathcal{B}; \Gamma \vdash x: \Omega_1 \vdash [x: \Sigma_1] \Omega_2}
\]

(application) \[
\frac{\mathcal{B}; \Gamma \vdash \Sigma_1; \text{type} \quad \mathcal{B}; \Gamma, x: \Sigma_1 \vdash \Omega_2; \text{type}}{
\mathcal{B}; \Gamma \vdash \Sigma_1 : \Omega_2}
\]

(conversion) \[
\frac{\mathcal{B}; \Gamma \vdash \Sigma_1; \text{type} \quad \mathcal{B}; \Gamma, x: \Sigma_1 \vdash \Omega_2; \text{type}}{
\mathcal{B}; \Gamma \vdash \Sigma_1 \equiv \Sigma_2; \text{type} \quad \mathcal{B}; \Gamma \vdash \Sigma_1 \equiv \Sigma_2}
\]

When using the parameter rule, we assume that \(\mathcal{B}; \Gamma \vdash \text{ok}\), 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), \(\emptyset; \emptyset \vdash \text{ok}\), so \((\emptyset; \text{prop}; \emptyset; \text{type}); \emptyset \vdash \text{ok}\) (book ext.: \(\text{pn1}\)).
2. By (parameters), \((1-1); \emptyset \vdash \text{prop}; \text{type}\). Therefore by (book ext.: \(\text{var1}\)),
   we have: \((1-1); (\emptyset; x, \text{--}, \text{prop}); \emptyset \vdash \text{ok}\).
3. By (context ext.), \((1-2); x; \text{prop} \vdash \text{ok}\).
   Therefore by (book ext.: \(\text{var1}\)), we have: \((1-2); (x; \text{prop}; y, \text{--}; \text{prop}); \emptyset \vdash \text{ok}\).
4. By two applications of (context ext.), \((1-3); x; \text{prop}, y; \text{prop} \vdash \text{prop}; \text{type}\).
   By (parameters), we have: \((1-3); x; \text{prop}, y; \text{prop} \vdash \text{prop}; \text{type}\).
   Therefore by (book ext.: \(\text{pn2}\)), we have: \((1-4); \emptyset \vdash \text{ok}\).

2.5 Definitional equality

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

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

\(^7\)This definition depends on the definition of derivability \(\vdash\) which in turn depends on the definition of \(=_{\beta d}\). The definitions of correct book, correct line, correct context, correct expression and \(=_{\beta d}\) 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 $\Sigma$ is a variable $x$, then $nf_d(\Sigma) \overset{\text{def}}{=} x$;

• If $\Sigma \equiv b(\Omega_1, \ldots, \Omega_n)$ and the normal forms of the $\Omega_i$s 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:
  
  o $\Xi_1 \equiv \varepsilon$. This case doesn’t occur, as $b \in \mathcal{C}$;
  
  o $\Xi_1 \equiv \text{PN}$. Then define $nf_d(\Sigma) \overset{\text{def}}{=} b(nf_d(\Omega_1), \ldots, nf_d(\Omega_n))$;
  
  o $\Xi_1$ is an expression. Then $\Xi_1$ 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 $nf_d(\Xi_1)$ has been defined. Define $nf_d(\Sigma) \overset{\text{def}}{=} nf_d(\Xi_1)[x_1, \ldots, x_n: = nf_d(\Omega_1), \ldots, nf_d(\Omega_n)]$;

• If $\Sigma \equiv [x: \Omega_1] \Omega_2$ then $nf_d(\Sigma) \overset{\text{def}}{=} [x: nf_d(\Omega_1)]nf_d(\Omega_2)$;

• If $\Sigma \equiv \langle \Omega_2 \rangle \Omega_1$ then $nf_d(\Sigma) \overset{\text{def}}{=} \langle nf_d(\Omega_2) \rangle nf_d(\Omega_1)$.

Write $\Sigma_1 =_d \Sigma_2$ if $nf_d(\Sigma_1) \equiv nf_d(\Sigma_2)^8$ and $=_d$ for the smallest equivalence relation containing $=_\beta$ and $=_d$.

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

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

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

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

$\Sigma$ is in $d$-normal form if for no expression $\Omega$, $\Sigma \rightarrow_\delta \Omega$. We define $\rightarrow_\delta, \rightarrow_\delta^+$ and $=_\delta$ as usual. Again, $\rightarrow_\delta$ 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 $A_1 \rightarrow_\delta B$ and $A_2 \rightarrow_\delta B$;

2. $nf_d(\Sigma)$ is the unique $d$-normal form of $\Sigma$;

3. $\Sigma =_\delta \Omega$ if and only if $\Sigma =_d \Omega$.

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

---

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

\(^9\)Definitional equality of expressions $\Sigma_1$ and $\Sigma_2$ depends on the book $\mathfrak{B}$, so we should write $=_{\beta\mathfrak{B}}$ instead of $=_{\beta\mathfrak{B}}$. As before, we leave out the subscript $\mathfrak{B}$ as long as no confusion arises.
Proof:

1. $\text{AUT-68}$ 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 $\Sigma \rightarrow_\delta n_f_\delta(\Sigma)$. By induction on the definition of $n_f_\delta(\Sigma)$ one shows that $n_f_\delta(\Sigma)$ is in $\delta$-normal form. The uniqueness of this normal form follows from the Church-Rosser property;

3. If $\Sigma =_\delta \Omega$ then by (1) there is $\Psi$ such that $\Sigma \rightarrow_\delta \Psi$ and $\Omega \rightarrow_\delta \Psi$. This means that the $\delta$-normal forms of $\Sigma$ and $\Omega$ are equal, so by (2), $n_f_\delta(\Sigma) \equiv n_f_\delta(\Omega)$. On the other hand, if $n_f_\delta(\Sigma) \equiv n_f_\delta(\Omega)$, then $\Sigma$ and $\Omega$ have the same $\delta$-normal forms (by (2)), so $\Sigma =_\delta \Omega$.

4. By 2, $\rightarrow_\delta$ is weakly normalising. Moreover, Definition 17 of $n_f_\delta(\Sigma)$ 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 $\Gamma$ is part of a context $\Gamma'$, notation $\Gamma \subseteq \Gamma'$, if all declarations $x:\alpha$ of $\Gamma$ are declarations in $\Gamma'$.

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

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

3 From $\text{AUT-68}$ towards a PTS $\lambda 68$

We want to give a description of $\text{AUT-68}$ 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 $\text{AUTOMATH}$. 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 Bentham Jutting in [2] of the systems $\text{AUT-68}$ and $\text{AUT-QE}$ 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 Bentham Jutting’s description of $\text{AUT-68}$ and $\text{AUT-QE}$.

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 $\text{AUTOMATH}$ 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 $\text{AUTOMATH}$, and it was shown that $\text{AUTOMATH}$ can be described very well within these PTSs with parameters.
To describe $\text{Aut-68}$ as a PTS without parameters (call it $\lambda 68$), we first translate the expressions of $\text{Aut-68}$ to typed $\lambda$-terms (note that the parameter mechanism of Definition 8 is replaced by repeated function application in PTSs):

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

- $\pi$ \text{def} \{ $x$ for $x \in V$; $b(\Sigma_1, \ldots, \Sigma_n)$ \text{def} \{ $\Sigma_1 \cdots \Sigma_n$; $(\Pi \Sigma) \text{def} \{ $\Pi \Sigma$\};
- $\text{type}$ \text{def} \{ $x : \Sigma$ if $[x] \Sigma$ has type $\text{type}$, \lambda x : \Sigma \Omega$ otherwise.

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

### 3.1 The choice of the correct formation ($\Pi$) rules and the parameter types $\Pi x : A. B$

As $\text{type} \equiv \ast$, Definition 15 clarifies which $\Pi$-rules are implied by the abstraction mechanism of $\text{Aut-68}$, the rule on the left translates into the rule on the right which is $\Pi$-rule $(\ast, \ast, \ast)$ ($\mathfrak{B}$ and $\Gamma$ are suitable translations of $\mathfrak{B}$ and $\Gamma$):

\[
\mathfrak{B}, \Gamma \vdash \Sigma_1 : \text{type} \quad \mathfrak{B}, \Gamma, x : \Sigma_1 \vdash \Omega : \text{type} \quad \mathfrak{B}, \Gamma \vdash \Sigma_1 : * \\
\mathfrak{B}, \Gamma, x : \Sigma_1 \Omega_1 : \text{type} \quad \mathfrak{B}, \Gamma \vdash \Sigma_1 : * \\
\mathfrak{B}, \Gamma \vdash (\Pi x : \Sigma_1 \Omega_1) : * \\
\]

It is, however, not immediately clear which $\Pi$-rules are induced by the parameter mechanism of $\text{Aut-68}$. Let $\Sigma \equiv b(\Sigma_1, \ldots, \Sigma_n)$ be a correct expression of type $\Omega$ with respect to a book $\mathfrak{B}$ and a context $\Gamma$. 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 $\Sigma_i$ is a correct expression with respect to $\mathfrak{B}$ and $\Gamma$, 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 $\Omega =_{sd} \Xi_2 [x_1, \ldots, x_n := \Sigma_1, \ldots, \Sigma_n]$. Now $\Sigma \equiv b(\Sigma_1, \ldots, \Sigma_n)$, and, assuming that we can derive in $\lambda 68$ that $\Sigma_i$ has type $\text{type} [x_1, \ldots, x_{i-1} := \Sigma_1, \ldots, \Sigma_{i-1}]$, it is not unreasonable to assign the type $\Pi x_1 : \text{type} \cdots \Pi x_n : \text{type} \Pi b : \text{type}$. We will abbreviate this last term by $\prod_{i=1}^n x_i : \text{type} \Pi b : \text{type}$. Then we can derive (using $n$ times the application rule that we will introduce for $\lambda 68$) that $\Sigma$ has type $\Pi \Sigma$ in $\lambda 68$.

It is important to notice that the type of $b$, $\prod_{i=1}^n x_i : \text{type} \Pi b$, does not necessarily have an equivalent in $\text{Aut-68}$, as in $\text{Aut-68}$ abstractions over $\text{type}$ are not allowed (only abstractions over expressions $\Sigma$ that have $\text{type}$ as type are possible — cf. Definition 15). In other words, the type of $b$, $\prod_{i=1}^n x_i : \text{type} \Pi b$, is not necessarily a first-class citizen of $\text{Aut-68}$ and should therefore have special treatment in $\lambda 68$. This is the reason to create a special sort $\Delta$, in which these types of $\text{Aut-68}$ constants and definitions are stored. This idea originates from van Benthem Jutting and was firstly presented in [2].
If we construct $\Pi x_n: \alpha_m \rightarrow \Xi_0$ from $\Xi_2$, 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 $\alpha_m$. As $\alpha_n \equiv \text{type}$ or $\alpha_n$ has type type, we must allow the possibilities $s_1 \equiv \ast$ and $s_1 \equiv \Box$. Similarly, $\Xi_2 \equiv \text{type}$ or $\Xi_2$ has type type, so we also allow $s_2 \equiv \ast$ and $s_2 \equiv \Box$. As we intended to store the new type in sort $\Delta$, we take $s_3 \equiv \Delta$.

For similar reasons, we introduce rules $(\ast, \Delta, \Delta)$ and $(\Box, \Delta, \Delta)$ to construct $\Pi_{i=1}^n x_i \alpha_i \rightarrow \Xi_2$ for $n > 1$. Hence, we have the $\Pi$-rules:

$(\ast, \ast, \ast); (\ast, \ast, \Delta); (\Box, \ast, \Delta); (\ast, \Box, \Delta); (\ast, \ast, \ast); (\Box, \ast, \ast)$.

We do not have rules of the form $(\Delta, s_2, s_3)$ or $(\ast_1, \Delta, s_3)$ with $s_3 \equiv \ast$ or $s_3 \equiv \Box$. So types of sort $\Delta$ cannot be used to construct types of other sorts.

In this way, we can keep the types of the $\lambda$-calculus part of AUT-68 separated from the types of the parameter mechanism: the last ones are stored in $\Delta$.

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

- The presence of both $(\ast, \ast, \ast)$ and $(\ast, \ast, \Delta)$ in the system stresses the fact that AUT-68 has two type mechanisms: one provided by the parameter mechanism and one by the $\lambda$-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 $\xi x:A.B$ for a type constructed by the parameter mechanism. Hence, we have for the constant $b$ above that $b : \xi_{i=1}^n x_i \alpha_i \rightarrow \Xi_2$.

As an additional advantage, the resulting system will maintain Unicity of Types. This would have been lost if we use rules $(\ast, \ast, \ast)$ and $(\ast, \ast, \Delta)$ without making this difference, as we can then derive both

$$\begin{align*}
\alpha : \ast & \vdash \alpha : \ast \\
\alpha : \ast & \vdash \alpha : \ast, x\alpha & \vdash \alpha : \ast
\end{align*}$$

and

$$\begin{align*}
\alpha : \ast & \vdash \alpha : \ast \\
\alpha : \ast & \vdash \alpha : \ast, x\alpha & \vdash \alpha : \ast
\end{align*}$$

- 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 $\Pi$-application. In AUT-68 with $\Pi$-application (call this system AUT-68II 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):

$$\begin{align*}
\emptyset; \Gamma \vdash \Sigma_1 : [x:\Omega_1] \Omega_2 & \quad \emptyset; \Gamma \vdash \Sigma_2 : \Omega_1 \\
\emptyset; \Gamma \vdash (\Sigma_2) \Sigma_1 : \Omega_2 [x=\Sigma_2] & \quad \emptyset; \Gamma \vdash (\Sigma_2) \Sigma_1 : (\Sigma_2) \Omega_2.
\end{align*}$$

So if we want to make a translation of AUT-68II, the application rule for $\Pi$-terms has to be different from the application rule for $\xi$-terms. Without distinction between $\Pi$-terms and $\xi$-terms, it would be impossible to amend the system to represent AUT-68II. Distinguishing between $\Pi$-terms and $\xi$-terms, it would be impossible to amend the system to represent AUT-68II.
\(\text{\(\Phi\)-terms makes it possible to obtain a translation of } \text{\(\text{Aut-68II}\)} \text{ from the translation of } \text{\(\text{Aut-68}\)} \text{ in a simple way.}

3.2 The different treatment of constants and variables

When we seek to translate the \text{\textit{\(\text{Aut-68}\)}} judgement \(\mathfrak{B}; \Gamma \vdash \Sigma : \Omega\) in \(\text{\(\lambda 68\)}\), we must pay attention to the translation of \(\mathfrak{B}\), as there is no equivalent of books in \(\text{\(\text{PTS}\)}\). Our solution is to store the information on identifiers of \(\mathfrak{B}\) in a \text{\(\text{PTS}\)-context}. Therefore, contexts of \(\text{\(\lambda 68\)}\) will have the form \(\Delta; \Gamma\). The left part \(\Delta\) 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 \(\Gamma\) 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 \text{\textit{\(\text{Aut-68}\)}} book, to which we add a line \((\Gamma; b: \text{\(\Phi\)}; \Sigma; \Xi)\). 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; \text{\(\Phi\)}\). In \(\text{\(\lambda 68\)}\) we can work as follows. Assume the information on constants in \(\mathfrak{B}\) has been translated into the left part \(\Delta\) of a \(\text{\(\lambda 68\)}\) context. We have (assuming that \(\text{\(\lambda 68\)}\) is a type system that behaves like \text{\textit{\(\text{Aut-68}\)}}), and writing \(\Gamma\) for the translation \(x_1: \alpha_1, \ldots, x_n: \alpha_n\) of \(\Gamma\): \(\Delta; \Gamma \vdash \Xi; \text{\(s\)}\) (\(s \equiv \ast\) if \(\mathfrak{B}; \Gamma \vdash \Xi; \text{\(\text{\(\Phi\)}\)}\); \(s \equiv \square\) if \(\Xi \equiv \text{\(\text{\(\Phi\)}\)}\)). Applying the \text{\textit{\(\text{\(\Phi\)}\)-formation}} rule \(n\) times, we obtain \(\Delta; \varnothing \vdash \Psi_1 \varnothing, \ldots, \Psi_n \varnothing : \Delta\) (if \(\Gamma\) is the empty context, then \(\Psi_1 \varnothing, \ldots, \Psi_n \varnothing\) has type \(\ast\) or \(\square\) instead of \(\Delta\)). We write \(\Psi_1\) for \(\prod_{i=1}^{n} x_i: \alpha_i\).

As \(\Psi_1\) 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 \(\Gamma\) of the original context has disappeared when we applied the \text{\textit{\(\text{\(\Phi\)}\)-formation}} rules, \(b: \Psi_1 \varnothing\) is automatically placed at the right-hand end of \(\Delta\): The conclusion of the start rule is \(\Delta; \varnothing \vdash \Psi_1 \varnothing, \ldots, \Psi_n \varnothing\).

Adding \(b: \Psi_1 \varnothing\) at the end of \(\Delta\) can be compared with adding the line \((\Gamma; b: \text{\(\Phi\)}; \Sigma; \Xi)\) at the end of \(\mathfrak{B}\). This process can be captured by the rule below where \(s_1 \in \{\ast, \square\}\) (compare: \(\Xi; \text{\(\text{\(\Phi\)}\)}\) or \(\Xi \equiv \text{\(\text{\(\Phi\)}\)}\)) and \(s_2 \in \{\ast, \square, \ast\} \) (usually, \(s_2 \equiv \Delta\); the cases \(s_2 \equiv \ast, \square\) only occur if \(\Gamma\) is empty):

\[
\begin{array}{c}
\Delta; \varnothing \vdash \Xi; s_1 \\
\Delta; \varnothing \vdash \Psi_1 \varnothing; s_2
\end{array}
\]

\[
\Delta; \varnothing \vdash \Psi_1 \varnothing, \ldots, \Psi_n \varnothing : \Delta; \varnothing \vdash \Psi_1 \varnothing, \ldots, \Psi_n \varnothing
\]

3.3 The definition system and the translation using \textit{\(\text{\(\Phi\)}\)}

A line \(x_1: \alpha_1, \ldots, x_n: \alpha_n; b: \Xi_1; \Xi_2\), in which \(b\) is a constant and \(\Xi_1 \in \mathfrak{E}\), 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 \(\text{\(\lambda 68\)}\), 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 \(b = (\lambda_{i=1}^{n} x_i: \alpha_i; \Xi_1): (\Xi_{i=1}^{n} x_i: \alpha_i; \Xi_2)\) in the context instead of only \(b; \Xi_{i=1}^{n} x_i: \alpha_i; \Xi_2\), and adding a \(\delta\)-reduction rule that unfolds the definition of \(b\): if \(b = (\lambda_{i=1}^{n} x_i: \alpha_i; \Xi_1): (\Xi_{i=1}^{n} x_i: \alpha_i; \Xi_2)\) \(\in \Delta\) then \(\Delta \vdash b \rightarrow s\). \(\lambda_{i=1}^{n} x_i: \alpha_i; \Xi_1\). Unfolding the definition of \(b\) in a term \(bX_1 \cdots X_n\) and applying \(\beta\)-reduction \(n\) times gives \(\Xi_2[x_1:=\Xi_1] \cdots [x_n:=\Xi_n]\). This procedure corresponds
exactly to the $\delta$-reduction $\Delta \vdash b(\Sigma_1, \ldots, \Sigma_n) \rightarrow_\delta \Xi_1[x_1, \ldots, x_n := \Sigma_1, \ldots, \Sigma_n]$ in $\text{AUT-68}^{11}$. This method, however, has disadvantages:

- In the $\text{AUT-68}$ line $(x_1; x_2; \ldots; x_n; a; \Xi_1; \Xi_2)$, $b(\Sigma_1, \ldots, \Sigma_n)$ has for equivalent in $\lambda^6$, $b\Sigma_1 \cdots \Sigma_m$. If $n > 0$, this $\lambda^6$-term has $B \equiv b\Sigma_1 \cdots \Sigma_m$ as a subterm for any $m < n$. But $B$ has no equivalent in $\text{AUT-68}$: only after $B$ is applied to suitable terms $\Sigma_{m+1}, \ldots, \Sigma_n$ the result $B\Sigma_{m+1} \cdots \Sigma_n$ has $b(\Sigma_1, \ldots, \Sigma_n)$ as its equivalent in $\text{AUT-68}$. Hence $B$ is not directly translatable into $\text{AUTMAT}$, 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 $\xi_{\lambda m+1} \cdot x_i; \xi_{\lambda m} \cdot \Xi_1$, of sort $\triangle$ (not * or $\Box$).

The method above allows to unfold the definition of $b$ already in $B$, because $b\Sigma_1 \cdots \Sigma_m$ can reduce to $(\lambda^6 \Sigma_1 \cdots \Sigma_{m+1} \cdot x_i; \xi_{\lambda m+1} \cdot \Xi_1) \Sigma_1 \cdots \Sigma_m$, and we can $\beta$-reduce this term $m$ times to $(\lambda^6 \Sigma_{m+1} \cdots \Sigma_n \cdot x_i; \xi_{\lambda m+1} \cdot \Xi_1) [x_j := \Sigma_{m+1}]$. It is more in line with $\text{AUT-68}$ to make such unfolding not possible before all $n$ arguments $\Sigma_1, \ldots, \Sigma_n$ have been applied to $b$, so only when the construction of the equivalent of $b(\Sigma_1, \ldots, \Sigma_n)$ has been completed;

- Moreover, $\lambda^6 \Sigma_1 \cdots \Sigma_m \cdot x_i; \xi_{\lambda m} \cdot \Xi_1$ does not necessarily have an equivalent in $\text{AUT-68}$. Take for example the constant $b$ in line $(\alpha; \text{type}; b; x; \alpha; x; \text{type}; \alpha)$.

Then $\lambda^6 \Sigma_1 \cdots \Sigma_m \cdot x_i; \xi_{\lambda m} \cdot \Xi_1 \equiv \lambda\alpha \cdot x; \alpha \cdot x$. Its equivalent in $\text{AUT-68}$ would be $[\alpha; \text{type}; \alpha; \alpha; x; \text{type}; \alpha; \alpha]$, but an abstraction $[\alpha; \text{type}; \alpha]$ cannot be made in $\text{AUT-68}$.\footnote{This explains why we do not incorporate $\lambda^6 \Sigma_1 \cdots \Sigma_m \cdot x_i; \xi_{\lambda m} \cdot \Xi_1$ as a citizen of $\lambda^6$.}

Therefore we choose a different translation. The line $(x_1; x_2; \ldots; x_n; a; \Xi_1; \Xi_2)$, where $\Xi_1 \in \mathcal{E}$, will be translated using $b := (\xi_{\lambda n} \cdot x_i; \xi_{\lambda n} \cdot \Xi_1) : (\xi_{\lambda n} \cdot x_i; \xi_{\lambda n} \cdot \Xi_2)$ instead of $b := (\lambda^6 \Sigma_1 \cdots \Sigma_m \cdot x_i; \xi_{\lambda m} \cdot \Xi_1) : (\xi_{\lambda m} \cdot x_i; \xi_{\lambda m} \cdot \Xi_2)$ in the left part of the translated context $\Delta$. A reduction rule $bX_1 \cdots X_n \rightarrow_\delta \Xi_1[x_1, \ldots, x_n := X_1, \ldots, X_n]$ is added for all terms $X_1, \ldots, X_n$. The symbol $\|/ is used instead of $\lambda$. This emphasises that, though both $\|/x; A$ and $\lambda x; A$ are abstractions, they are not the same kind of abstraction.

4 $\lambda^6$

Here, we give $\lambda^6$, show that it has the desirable properties of PTSs and that it is the PTS version of $\text{AUT-68}$.

**Definition 24 ($\lambda^6$)**

1. Let $S$ is the set of sorts $\{*, \Box, \triangle\}$. Terms of $\lambda^6$ are given by $T := \mathcal{V} \mid C \mid S \mid T \cdot T \mid \lambda \mathcal{V} : T \cdot T \mid \|/ \mathcal{V} : T \cdot T \mid \|/ \mathcal{V} : T \cdot T$. Free variables $\mathcal{FV}(T)$ and "free" constants $\mathcal{F}(T)$ of term $T$ are defined as usual;

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

12This situation compares to that of Section 3.1, where we found that the type of $b$ is not necessarily a first-class citizen of $\text{AUT-68}$. There, we could not avoid that the type of $b$ became a citizen of $\lambda^6$ (though we made it a second-class citizen by storing it in the sort $\triangle$).
2. We define the notion of context inductively:

- \( \emptyset ; \emptyset \) is a context; \( \text{DOM}(\emptyset ; \emptyset ) = \emptyset \);

- If \( \Delta_1; \Gamma \) is a context, \( x \in \mathcal{V} \), \( x \) does not occur in \( \Delta_1; \Gamma \) and \( A \in \mathcal{T} \), then \( \Delta_1; \Gamma, x:A \) is a context (\( x \) is a newly introduced variable); \( \text{DOM}(\Delta_1; \Gamma, x:A) = \text{DOM}(\Delta_1; \Gamma) \cup \{ x \} \);

- If \( \Delta_1; \Gamma \) is a context, \( b \in \mathcal{C} \), \( b \) does not occur in \( \Delta_1; \Gamma \) and \( A \in \mathcal{T} \) then \( \Delta, b:A; \Gamma \) is a context (in this case \( b \) is a primitive constant); \( \text{DOM}(\Delta, b:A; \Gamma) = \text{DOM}(\Delta_1; \Gamma) \cup \{ b \} \);

- If \( \Delta_1; \Gamma \) is a context, \( b \in \mathcal{C} \), \( b \) does not occur in \( \Delta_1; \Gamma \), \( A \in \mathcal{T} \), and \( T \in \mathcal{T} \), then \( \Delta, b:=T; A; \Gamma \) is a context (in this case \( b \) is a defined constant); \( \text{DOM}(\Delta, b:=T; A; \Gamma) = \text{DOM}(\Delta_1; \Gamma) \cup \{ b \} \).

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 \( \text{PRIMCONS}(\Delta_1; \Gamma) = \{ b \in \text{DOM}(\Delta_1; \Gamma) \mid b \) is a primitive constant \}; \)

\( \text{DEFCONS}(\Delta_1; \Gamma) = \{ b \in \text{DOM}(\Delta_1; \Gamma) \mid b \) is a defined constant \}; and

\( \text{FV}(\Delta_1; \Gamma) = \text{DOM}(\Delta_1; \Gamma) \).

3. We define \( \delta \)-reduction on terms. Let \( \Delta \) be the left part of a context.

If \( (b := (\forall_{x_1, .., x_n} \in \mathcal{C}, A, T) : (\forall_{y_1, .., y_m} \in \mathcal{C}, B_1, B_2)) \) \( \in \Delta \) and \( B \) is not \( \forall y; B_1, B_2 \), then \( \Delta \vdash b x_1 \cdot \cdot \cdot x_n \rightarrow_\delta T[x_1, \ldots, x_n := X_1, \ldots, X_n] \) for all \( X_1, \ldots, X_n \in \mathcal{T} \).

We also have the usual compatibility rules on \( \delta \)-reduction. We use notations like \( \rightarrow_\delta, \rightarrow^+_\delta, =_\delta \) as usual. If no confusion about which \( \Delta \) occurs, we simply write \( \Delta \vdash b x_1 \cdot \cdot \cdot x_n \rightarrow_\delta T[x_1, \ldots, x_n := X_1, \ldots, X_n] \).

4. We use the usual notion of \( \beta \)-reduction;

5. Judgements in \( \lambda 68 \) have the form \( \Delta_1; \Gamma \vdash A : B \), where \( \Delta_1; \Gamma \) is a context and \( A \) and \( B \) are terms. In the case that a judgement \( \Delta_1; \Gamma \vdash A : B \) is derivable according to the rules below, \( \Delta_1; \Gamma \) is a legal context and \( A \) and \( B \) are legal terms. We write \( \Delta_1; \Gamma \vdash A : B : C \) if both \( \Delta_1; \Gamma \vdash A : B \) and \( \Delta_1; \Gamma \vdash B : C \) 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 \( \Delta_1; \Gamma \vdash_{\lambda 68} A : B \) instead of \( \Delta_1; \Gamma \vdash A : B \) if the latter gives rise to confusion.

Note that there is no rule \( (\beta) \). This is because we do not want terms like \( \forall x : A. B \) 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 \( b := (\forall_{x_1, .., x_n} \in \mathcal{C}, A, U) : B \in \Delta_2 \) and \( U \not\equiv \forall y: B, A' \) then \( \Delta_1 \vdash b x_1 \cdot \cdot \cdot x_n =_\beta U \).
(Axiom) \[
\vdash * : \Box
\]

(Start : v) \[
\frac{\Delta ; \Gamma \vdash A : s \quad \Delta ; \Gamma , x : A \vdash x : A}{\Delta ; \Gamma , x : A \vdash \lambda x. A : s} \quad s \equiv *, \Box
\]

(Start : pc) \[
\frac{\Delta ; \Gamma \vdash T : B : s_1 \quad \Delta ; \Gamma , \phi : B \vdash \psi : B}{\Delta ; \Gamma ; \phi \equiv (\Delta ; \Gamma , T) : \psi \vdash \phi \equiv \Delta ; \Gamma , T : s_1 \quad s_1 \equiv *, \Box}
\]

(Start : dc) \[
\frac{\Delta ; \Gamma \vdash T : B : s_1 \quad \Delta ; \Gamma , \phi : B \vdash \psi : B}{\Delta ; \Gamma , \phi \equiv (\Delta ; \Gamma , T) : \psi \vdash \phi \equiv \Delta ; \Gamma , T : s_1 \quad s_1 \equiv *, \Box}
\]

(Weak : v) \[
\frac{\Delta ; \Gamma , x : A \vdash M : N}{\Delta ; \Gamma \vdash \lambda x. A : s} \quad s \equiv *, \Box
\]

(Weak : pc) \[
\frac{\Delta ; \Gamma \vdash M : N \quad \Delta ; \Gamma , \phi : B \vdash \psi : B}{\Delta ; \Gamma , \phi \equiv (\Delta ; \Gamma , T) : \psi \vdash \phi \equiv \Delta ; \Gamma , T : s_1 \quad s_1 \equiv *, \Box}
\]

(Weak : dc) \[
\frac{\Delta ; \Gamma \vdash M : N \quad \Delta ; \Gamma , \phi : B \vdash \psi : B}{\Delta ; \Gamma , \phi \equiv (\Delta ; \Gamma , T) : \psi \vdash \phi \equiv \Delta ; \Gamma , T : s_1 \quad s_1 \equiv *, \Box}
\]

(Π - form) \[
\frac{\Delta \vdash A : *}{\Delta \vdash (\forall x. A) : *}
\]

(∀ - form) \[
\frac{\Delta ; \Gamma \vdash \forall x. A : *}{\Delta ; \Gamma \vdash (\forall x. A) : *}
\]

(λ) \[
\frac{\Delta ; \Gamma \vdash \forall x. A : *}{\Delta ; \Gamma \vdash (\forall x. A) : *}
\]

(App1) \[
\frac{\Delta ; \Gamma \vdash M \equiv (\forall x. A) : (\forall x. A) \quad \Delta ; \Gamma \vdash N : A}{\Delta ; \Gamma \vdash MN : B[x = N]}
\]

(App2) \[
\frac{\Delta ; \Gamma \vdash M \equiv (\forall x. A) \quad \Delta ; \Gamma \vdash N : A}{\Delta ; \Gamma \vdash MN : B[x = N]}
\]

(Conv) \[
\frac{\Delta ; \Gamma \vdash M : A \quad \Delta ; \Gamma \vdash B : s \quad \Delta \vdash A = s B}{\Delta ; \Gamma \vdash M : B}
\]

Figure 4: Rules of \(\lambda\text{68}\)
Many properties for PTs hold for \(\lambda S\) and can be proved by the same methods as for PTs. 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 \(\Delta \equiv b_1 \cdot B_1, \ldots, b_m \cdot B_m\) (in \(\Delta\), also expressions \(b_i = T_i : B_i\) may occur, but for uniformity of notation we leave out the :=-part); let \(\Gamma \equiv x_1 : A_1, \ldots, x_n : A_n\) and \(\Delta; \Gamma \vdash M : N\). Then:

- The \(b_1, \ldots, b_m \in C\) and \(x_1, \ldots, x_n \in V\) are all distinct;
- \(FC(M), FC(N) \subseteq \{b_1, \ldots, b_m\}\); \(FV(M), FV(N) \subseteq \{x_1, \ldots, x_n\}\);
- \(b_1 \cdot B_1, \ldots, b_{i-1} \cdot B_{i-1}, b_i \cdot B_i \vdash s_i\) for \(s_i \in \{*, \Box, \Delta\}\); 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 \(\Delta; \Gamma \vdash * : \Box, \text{ and if } b : A \in \Delta; \Gamma, \text{ or } c : T : A \in \Delta, \text{ then } \Delta; \Gamma \vdash c : A\).

- **(Definition)** Assume \(\Delta_1, b : \left(\gamma_1^{n-1} x_i : A_i ; T\right) ; \left(\xi_1^{n-1} x_i : A_i ; B\right)\) and \(\Delta_2; \Gamma \vdash M : N\), where \(B\) is not of the form \(\eta y : B_1 . B_2\). Then \(\Delta_1; x_1 : A_1, \ldots, x_n : A_n \vdash T : B : s\) for \(s \in \{*, \Box\}\).

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 \(\Delta_1 \equiv b_1 : *, b_2 : *, b_3 : = b_1 : *\) and \(\Delta_2 \equiv b_1 : *, b_2 : *, b_3 : = b_2 : *\). Let \(\Gamma_1 \equiv \Gamma_2 \equiv x_3 : b_3\). 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 \(\Delta_2; \Gamma_2 \vdash x_3 : b_2\) (because \(\Delta_2; \Gamma_2 \vdash x : b_3\) and according to \(\Delta_2, b_3 =_{\beta_1} b_2\), so we can use the conversion rule). But we cannot derive \(\Delta_1; \Gamma_1 \vdash x_3 : b_2\) (because \(b_3\) and \(b_2\) are not definitionally equal according to \(\Delta_1\)).

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 \(b = \left(\gamma_1^{n-1} x_i : A_i ; U\right) \in \Delta_2\) and \(U \neq \gamma y : B\). Then \(\Delta_1 \vdash bx_1 \cdot \ldots \cdot x_n =_{\beta_0} U\).
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 \( \Delta; \Gamma, x : A, \Gamma_2 \vdash B : C \) and \( \Delta; \Gamma \vdash D : A \) 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 \( \exists s \in \{*, \square\} \) and \( B =_{\beta_\delta} C \) such that \( \Delta; \Gamma \vdash B : s \) and \( x : B \in \Gamma \);

- If \( b \in C \) and \( \Delta; \Gamma \vdash b : C \) then \( \exists s \in \mathcal{S} \) and \( B =_{\beta_\delta} C \) such that \( \Delta; \Gamma \vdash B : s \), and either \( b : B \in \Delta \) or \( \exists T \) such that \( \langle b := T : B \rangle \in \Delta \);

- If \( s \in \mathcal{S} \) and \( \Delta; \Gamma \vdash s : C \) then \( s \equiv * \) and \( C =_{\beta_\delta} \square \);

- If \( \Delta; \Gamma \vdash MN : C \) then \( \exists A, B \) such that \( \Delta; \Gamma \vdash (\Pi x : A. B) \) or \( \Delta; \Gamma \vdash M : (\lambda x : A. B), \) 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 \( \Delta; \Gamma \vdash (\lambda x : A. B) : C \) then \( C =_{\beta_\delta} \Delta, \Delta; \Gamma \vdash A : s_1 \) for \( s_1 \in \{*, \square\} \), and \( \Delta; \Gamma, x : A \vdash B ; s_2 \) for \( s_2 \in \{*, \square, \Delta\} \).

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 \( s \in \mathcal{S} \) 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 \( \Delta; \Gamma \vdash A : (\lambda x : B_1, B_2) \) then \( \Delta; \Gamma \vdash B_1 : s_1 \) for \( s_1 \in \{*, \square\} \); 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 \( \rightarrow_\beta \), \( \rightarrow_{\lambda_\delta} \) and \( \rightarrow_{\beta_\delta} \) and as \( \delta \)-reduction also depends on books, we first have to give a translation of \( \text{Aut-68} \)-books and \( \text{Aut-contexts} \) to \( \lambda_68 \)-contexts:

Definition 33  * Let \( \Gamma \) be a \( \text{Aut-68} \)-context \( x_1 : \alpha_1, \ldots, x_n : \alpha_n \). Then \( \Gamma \defeq x_1 ; x_1^{-1} ; \ldots ; x_n ; x_n^{-1} \).

- Let \( \mathcal{B} \) be a book. We define the left part \( \overline{\mathcal{B}} \) of a context in \( \lambda_68 \) as:

  \[ \overline{\mathcal{B}} \defeq \emptyset ; \]

  \[ \overline{\mathcal{B}}, (\Gamma ; x ; \Delta) \defeq \overline{\mathcal{B}} ; \]

  \[ \overline{\mathcal{B}}, (\Gamma ; x ; \Sigma ; \Omega) \defeq \overline{\mathcal{B}}, b : \Gamma ; \Omega ; \]

  \[ \overline{\mathcal{B}}, (\Gamma ; b ; \Sigma ; \Omega) \defeq \overline{\mathcal{B}}, b : \Gamma ; \Sigma ; \Omega ; \]
Example 34  The translation of the Automath book of Example 13 into λ68 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 B.

1. Σ  \rightarrow_{\beta} \Sigma' if and only if \Sigma  \rightarrow_{\beta} \Sigma';
2. B ⊢_{AT_{\lambda68}} Σ  \rightarrow_{\delta} \Sigma' if and only if \overline{B} ⊢_{\lambda68} \overline{Σ}  \rightarrow_{\delta} \overline{Σ'}.

Proof: An easy induction on the structure of Σ.

The Church-Rosser property of \rightarrow_{\beta_{68}} (Theorem 44) will be proved by Parallel Reduction \rightarrow_{p_{68}}, 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_{p_{68}} on T:

- For x ∈ V, Δ ⊢ x \Rightarrow_{p_{68}} x;
- For b ∈ C, Δ ⊢ b \Rightarrow_{p_{68}} b;
- For s ∈ S, Δ ⊢ s \Rightarrow_{p_{68}} s;
- If Δ ⊢ P := Q \Rightarrow_{p_{68}} \Pi_{x : P′} Q′, then
  o Δ ⊢ \Pi_{x : P} Q \Rightarrow_{p_{68}} \Pi_{x : P′} Q′;
  o Δ ⊢ \Pi_{x : P} Q \Rightarrow_{p_{68}} \Pi_{x : P′} Q′;
- If Δ ⊢ Q \Rightarrow_{p_{68}} Q′ and Δ ⊢ R \Rightarrow_{p_{68}} R′, then Δ ⊢ (\lambda x : P) Q \Rightarrow_{p_{68}} (\lambda x : P) Q′ [x := R′];
- If \overline{b} = \langle \sum_{i=1}^{n} x_{i} : A_{i} T_{i} \rangle : \langle \sum_{i=1}^{n} x_{i} : A_{i} U \rangle ∈ Δ, the term T is not of the form \sum_{i=1}^{n} x_{i} : T_{1} \cdot T_{2}, Δ ⊢ T \Rightarrow_{p_{68}} T′ and Δ ⊢ M_{i} \Rightarrow_{p_{68}} M_{i}′ for i = 1, . . . , n, then Δ ⊢ b M_{1} · · · M_{n} \Rightarrow_{p_{68}} T′[x_{1}, . . . , x_{n} := M_{1}′, . . . , M_{n}′].

Some elementary properties of \Rightarrow_{p_{68}} are:

Lemma 37 (Properties of \Rightarrow_{p_{68}}) Let Δ be the left part of a context. Then:

1. Δ ⊢ M \Rightarrow_{p_{68}} M;
2. If Δ ⊢ M \Rightarrow_{p_{68}} M′ then Δ ⊢ M \Rightarrow_{p_{68}} M′;
3. If Δ ⊢ M \Rightarrow_{p_{68}} M′ then Δ ⊢ M \Rightarrow_{p_{68}} M′.

Proof: All proofs can be given by induction on the structure of M.
By Lemma 37, \( \rightarrow_{\beta} \) (the reflexive and transitive closure of \( \rightarrow_{\beta} \)) in \( \Delta \) is the same relation as the reflexive and transitive closure of \( \rightarrow_{\beta} \) in \( \Delta \). Therefore, if we want to prove Church-Rosser \( \Rightarrow_{\beta} \), it suffices to prove the Diamond Property for \( \Rightarrow_{\beta} \). We first make some preliminary definitions and remarks:

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

**Proof:** Induction on the structure of \( M \).

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

**Proof:** By induction on the length of \( \Delta \) and by induction on the definition of \( \Delta \vdash M \Rightarrow_{\beta} N \). All cases in the definition of \( \Delta \vdash M \Rightarrow_{\beta} N \) follow directly from IH for \( \Delta \vdash M \Rightarrow_{\beta} N \), except for the case \( b M_1 \cdots M_n \Rightarrow_{\beta} T'[x_1, \ldots, x_n := M'_1, \ldots, M'_n] \). As \( FC(M) \subseteq DOm(\Delta) \), we have \( b \in DOm(\Delta) \).

Write \( \Delta \equiv \Delta_1, b := (\xi_{i=1}^n x_i : A_i, T); (\xi'_{i=1}^n x'_i : A'_i, U), \Delta_2 \).

- Notice that \( T \) is typable in \( \Delta_1; x_1 : A_1, \ldots, x_n : A_n \) (Definition Lemma). By the Free Variable Lemma: \( FC(T) \subseteq DOm(\Delta_1) \). By IH on the length of \( \Delta \) we have \( \Delta_1 \vdash T \Rightarrow_{\beta} T' \) iff \( \Delta \vdash T \Rightarrow_{\beta} T' \), and \( \Delta_1 \vdash T \Rightarrow_{\beta} T' \) iff \( \Delta, \Delta' \vdash T \Rightarrow_{\beta} T' \);

- We conclude: \( \Delta \vdash T \Rightarrow_{\beta} T' \) iff \( \Delta, \Delta' \vdash T \Rightarrow_{\beta} T' \);

- By IH on the definition of \( \Delta \vdash M \Rightarrow_{\beta} N \), we have \( \Delta \vdash M_i \Rightarrow_{\beta} M'_i \) iff \( \Delta, \Delta' \vdash M_i \Rightarrow_{\beta} M'_i \);

- Note that \( b := (\xi_{i=1}^n x_i : A_i, T); (\xi'_{i=1}^n x'_i : A'_i, U) \) is an element of both \( \Delta, \Delta' \) and \( \Delta \). Moreover, \( b \not\in DOm(\Delta') \) (as \( \Delta, \Delta' \) is the left part of a legal context). Hence \( \Delta \vdash b M_1 \cdots M_n \Rightarrow_{\beta} N \) iff \( \Delta, \Delta' \vdash b M_1 \cdots M_n \Rightarrow_{\beta} N \).

For left parts \( \Delta \) of contexts and for \( M \in T \) with \( FC(M) \subseteq DOm(\Delta) \), we define a term \( M^\Delta \). In \( M^\Delta \), all \( \beta \)-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 \( \delta \)-redexes are contracted. We will show that \( \Delta \vdash N \Rightarrow_{\beta} M^\Delta \) for any \( N \) with \( \Delta \vdash M \Rightarrow_{\beta} N \); so \( M^\Delta \) helps us to show the Diamond Property for \( \Rightarrow_{\beta} \).

**Definition 40** We define \( M^\Delta \) for any left part \( \Delta \) of a context and any \( M \in T \) such that \( FC(M) \subseteq DOm(\Delta) \). The definition of \( M^\Delta \) is by induction on the length of \( \Delta \). So assume \( M^\Delta \) has been defined for contexts \( \Delta' \) shorter than \( \Delta \).

We use induction on the structure of \( M \):

- \( x^\Delta \overset{\text{def}}{=} x \) for any \( x \in \mathcal{V} \);
- \( s^\Delta \overset{\text{def}}{=} s \) for any \( s \in \mathcal{S} \);

- \( M \equiv b \). Distinguish:
  
  - \( b^\Delta \overset{\text{def}}{=} b \) for any \( b \in \text{PRIMCONS} (\Delta) \);
  
  - \( b^\Delta \overset{\text{def}}{=} b \) for any \( b \in \text{DEFCONS} (\Delta) \) that is not a \( \delta \)-redex;
- If $b \in \text{DEFCONS}(\Delta)$ is a $\delta$-redex, then $\Delta \equiv \Delta,b;=T:U,\Delta_2$, where $T \not\equiv \emptyset_y T_1,T_2$. By the Definition Lemma, $\Delta_1;\vdash T : U$, so we can assume that $T^{\Delta_1}$ has already been defined. Then $b^\Delta \overset{\text{def}}{=} T^{\Delta_1};$

- $(\lambda x.P.Q)^\Delta \overset{\text{def}}{=} \lambda x.P^\Delta \cdot Q^\Delta$;  

- $(\Pi x;P.Q)^\Delta \overset{\text{def}}{=} \Pi x;P^\Delta \cdot Q^\Delta$;

- $M$ is an application term. We distinguish three possibilities:

- $M \equiv PQ$ is not a $\beta\delta$-redex. Then we define $M^\Delta \overset{\text{def}}{=} P^\Delta \cdot Q^\Delta$;

- $M$ is a $\beta$-redex $(\lambda x;P.Q)R$. We define $M^\Delta \overset{\text{def}}{=} Q^\Delta [x:=R^\Delta]$;

- $M$ is a $\delta$-redex $bM_1 \cdots M_n$, and for $T$ is not of the form $\emptyset_y T_1,T_2$, $\Delta$ is $\Delta_1,b;=(\emptyset_n^\Delta x_i:A_i,T);\emptyset_n^\Delta x_i:A_i,U) \cdot \Delta_2$. So $\Delta_1;x_1:A_1,\ldots,x_n:A_n \vdash T : U$ (by the Definition Lemma) and we can assume that $T^{\Delta_1}$ has already been defined. Then $M^\Delta \overset{\text{def}}{=} T^{\Delta_1};[x_1,\ldots,x_n:=M_1^\Delta,\ldots,M_n^\Delta]$.

\textbf{Lemma 41} Let $\Delta$ be the left part of a legal context. Then $\Delta \vdash M \Rightarrow_{\beta\delta} M^\Delta$ for all $M$ with $\text{FC}(M) \subseteq \text{DOM}(\Delta)$.

\textbf{Proof:} By induction on the definition of $M^\Delta$. 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 $\delta$-redex. Write $\Delta \equiv \Delta_1,b;=(\emptyset_n^\Delta x_i:A_i,T);\emptyset_n^\Delta x_i:A_i,U) \cdot \Delta_2$, 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, $\text{FC}(T) \subseteq \text{DOM}(\Delta_1)$. By Lemma 39, $\Delta \vdash T \Rightarrow_{\beta\delta} T^{\Delta_1}$. So $\Delta \vdash bM_1 \cdots M_n \Rightarrow_{\beta\delta} T^{\Delta_1};[x_1,\ldots,x_n:=M_1^\Delta,\ldots,M_n^\Delta]$. 

\textbf{Theorem 42} Let $\Delta$ be the left part of a legal context. Assume $\text{FC}(M) \subseteq \text{DOM}(\Delta)$. If $\Delta \vdash M \Rightarrow_{\beta\delta} N$ then $\Delta \vdash N \Rightarrow_{\beta\delta} M^\Delta$.

\textbf{Proof:} Induction on the the definition of $M^\Delta$.

- $M \equiv x$. Then $N \equiv x$ and $M^\Delta \equiv x$;

- $M \equiv b$. Distinguish:

  - $b \in \text{PRIMCONS}(\Delta)$. Then $N \equiv b$ and $M^\Delta \equiv b$;

  - $b \in \text{DEFCONS}(\Delta)$, but $b$ is not a $\delta$-redex. Then $N \equiv b$ and $M^\Delta \equiv b$;

  - $b \in \text{DEFCONS}(\Delta)$, and $\Delta \equiv \Delta_1,b;=T:U,\Delta_2$, and $T \not\equiv \emptyset_y T_1,T_2$. 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 $\Delta \vdash \lambda x;P'.Q' \Rightarrow_{\beta\delta} \lambda x;P^\Delta \cdot Q^\Delta$. The cases $M \equiv \Pi x;P.Q$, $M \equiv \xi x;P.Q$, and $M \equiv PQ$ where $PQ$ is not a $\beta\delta$-redex, are proved similarly;
\* M is an application term (and is either a \( \beta \) or a \( \delta \)-redex). Distinguish:

- \( M \) is a \( \beta \)-redex, \( M \equiv (\lambda x: P \cdot Q)R \). Distinguish:
  * \( N \equiv (\lambda x: P \cdot Q')R' \) 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, \( \Delta \vdash Q \Rightarrow_{\beta_\Delta} Q' \) and \( \Delta \vdash R \Rightarrow_{\beta_\Delta} R' \). 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 \( \delta \)-redex, \( M \equiv b M_1 \cdots M_n \), and for \( T \not\equiv \frac{y}{y} T_1, T_2 \), we have
  \[ \Delta \equiv \Delta_1, b := (\sum_{i=1}^n x_i:A_i, T), (\sum_{i=1}^n x_i:A_i, U), \Delta_2. \]
  * \( N \equiv b M'_1 \cdots M'_n \) for \( M_i \) with \( \Delta \vdash M_i \Rightarrow_{\beta_\Delta} M_i' \). By induction, we have \( \Delta \vdash M'_i \Rightarrow_{\beta_\Delta} M^{\Delta}_i \). 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, \( FC(T) \subseteq \text{DOM}(\Delta_1) \). 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 \( \Delta \vdash N \Rightarrow_{\beta_\Delta} T^{\Delta_1}; \)
  * \( N \equiv T'[x_1, \ldots, x_n:=M'_1, \ldots, M'_n] \) 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, \( FC(T) \subseteq \text{DOM}(\Delta_1) \). By Lemma 39, \( \Delta_1 \vdash T \Rightarrow_{\beta_\Delta} T^{\Delta_1} \). By IH on \( T \), \( \Delta_1 \vdash T' \Rightarrow_{\beta_\Delta} T^{\Delta_1} \). As \( \Delta_1 \vdash T \Rightarrow_{\beta_\Delta} T^{\Delta_1} \), \( FC([T]) \subseteq \text{DOM}(\Delta_1) \), so by Lemma 39, \( \Delta \vdash T' \Rightarrow_{\beta_\Delta} T^{\Delta_1} \). By IH, also \( \Delta \vdash M'_i \Rightarrow_{\beta_\Delta} M^{\Delta}_i \). Repeatedly applying Lemma 38, we find\(^{13}\)
  \[ \Delta \vdash T'[x_1, \ldots, x_n:=M'_1, \ldots, M'_n] \Rightarrow_{\beta_\Delta} T^{\Delta_1}; \]

Corollary 43 (Diamond Property for \( \Rightarrow_{\beta_\Delta} \)) Let \( \Delta \) 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 \( \Delta \) be the left part of a context in which \( M \) is typable. If \( \Delta \vdash M \Rightarrow_{\beta_\Delta} N_1 \) 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: 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. \( A \Rightarrow_{\beta} A' \) then \( \Delta; \Gamma \vdash A' : B \).
3. If \( A \Rightarrow_{\beta_\delta} A' \) 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 \( T^{\Delta_1}[x_1, \ldots, x_n:=M^{\Delta_1}_1, \ldots, M^{\Delta_1}_n] \equiv T^{\Delta_1}[x_1:=M^{\Delta_1}_1] \cdots [x_n:=M^{\Delta_1}_n] \). This is correct as we can assume that the \( x_i \) do not occur in the \( M'_j \) and \( M^{\Delta}_j \).
**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 $\Gamma \equiv \Gamma_1,x:A_{\delta},\Gamma_2$, 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:

$$
\Delta;\Gamma \vdash A:B \quad \text{and} \quad \Delta \vdash A \to_{\delta} A' \quad \Rightarrow \quad \Delta;\Gamma \vdash A':B
$$

$$
\Delta;\Gamma \vdash A:B \quad \text{and} \quad \Delta;\Gamma \to_{\delta} \Delta';\Gamma \quad \Rightarrow \quad \Delta';\Gamma \vdash A:B
$$

$$
\Delta;\Gamma \vdash A:B \quad \text{and} \quad \Delta;\Gamma \to_{\delta} \Delta';\Gamma' \quad \Rightarrow \quad \Delta';\Gamma' \vdash A:B,
$$

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:

$$
\Delta \equiv \Delta_1,b = \left(\frac{n}{3} x_i:A_i.T\right) \cdot \left(\frac{m}{3} x_i:A_i.B\right), \Delta_2 \quad (1)
$$

with $B \neq \emptyset, B_i, B_2$, and that the conclusion of the 2nd application rule is

$$
\Delta;\Gamma \vdash b M_1 \cdots M_n : K_n
$$

for some $K_n$, and therefore $\Delta \vdash b M_1 \cdots M_n \to_{\delta} T[x_i := M_i]_i^{n+1}$. We must prove:

$$
\Delta;\Gamma \vdash T[x_i := M_i]_i^{n+1} : K_n.
$$

We do this in two steps.

1. We analyse the structure of $K_n$, and derive that $\Delta \vdash K_n = \_\delta B[x_i := M_i]_i^{n+1}$

2. We show that $\Delta;\Gamma \vdash T[x_i := M_i]_i^{n+1} : B[x_i := M_i]_i^{n+1}$.

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

$$
\Delta;\Gamma \vdash b M_1 \cdots M_i - 1 : (\emptyset x_i : L_i) \quad (3)
$$

$$
\Delta;\Gamma \vdash M_i : L_i \quad (4)
$$

$$
\Delta \vdash k_i = \_\delta K_i[x_i := M_i] \quad (5)
$$

$$
\Delta \vdash k_{i-1} = \_\delta \emptyset x_i : L_i, K_i' \quad (6)
$$

We end with $\Delta;\Gamma \vdash b : (\emptyset x_i : L_i, K_i')$. By (1) and Generation: $\Delta \vdash \emptyset x_i : L_i, K_i' = \_\delta \emptyset x_i : A_j : B$. By Church-Rosser we have $L_1 = \_\delta A_1$ and

$$
\Delta \vdash k_i' = \_\delta \frac{n}{2} x_j : A_j : B. \quad (7)
$$

Hence $\Delta \vdash k_2 : L_2, K_2' = (\_\delta K_1 \equiv (\_\delta \emptyset x_i : A_j : B)[x_1 := M_1] \equiv \emptyset x_i : A_j[x_1 := M_1] B[x_1 := M_1]$; by the Church-Rosser Theorem we have $L_2 = \_\delta A_2[x_1 := M_1]$. Proceeding in this way, we obtain for $i = 1, \ldots, n$:

$$
\Delta \vdash L_i = \_\delta A_i[x_i := M_i]_i^{n+1}. \quad (8)
$$

$$
\Delta \vdash K_i' = \_\delta \frac{n}{2} x_j : A_j[x_k := M_k]_k^{n+1} B[x_k := M_k]_k^{n+1}. \quad (9)
$$

$$
\Delta \vdash K_i = \_\delta \frac{n}{2} x_j : A_j[x_k := M_k]_k^{n+1} B[x_k := M_k]_k^{n+1}. \quad (10)
$$

32
In particular,
\[ \Delta \vdash K_n =_\beta B[x_i := M_i]_{i=1}^n. \tag{9} \]

**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: \( \Delta_1 ; x_1 : A_1, \ldots, x_{n-1} : A_{i-1} \vdash A_i \vdash s_i \) for sorts \( s_i \in S \). Hence:

\[ \Delta; \Gamma \vdash A_1 : s_1 \]  (Thinning Lemma);

\[ \Delta; \Gamma, x_1 : A_1 \text{ is legal} \]  (Start Rule);

\[ \Delta; \Gamma, x_1 : A_1 \vdash A_2 : s_2 \]  (Thinning Lemma);

\[ \Delta; \Gamma, x_1 : A_1, x_2 : A_2 \text{ is legal} \]  (Start Rule);

\[ \vdots \]

\[ \Delta; \Gamma, x_1 : A_1, \ldots, x_n : A_n \text{ is legal.} \]  (Start Rule).

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: \( \Delta; \Gamma, x_2 : A_2 [x_1 := M_1], \ldots, x_n : A_n [x_1 := M_1] \vdash T [x_i := M_i] : B[x_i := M_i] \); and \( \Delta; \Gamma \vdash A_2 [x_1 := M_1] : s_2 \).

As \( \Delta; \Gamma \vdash M_2 : L_2 \) (4) and \( \Delta \vdash A_2 [x_1 := M_1] =_\beta s_2 \) (8) we have by conversion \( \Delta; \Gamma \vdash M_2 : A_2 [x_1 := M_1] \), and again by the Substitution Lemma:

\[ \Delta; \Gamma, x_3 : A_3 [x_i := M_i]_{i=1}^2, \ldots, x_n : A_n [x_i := M_i]_{i=1}^2 \vdash T [x_i := M_i]_{i=1}^n : B[x_i := M_i]_{i=1}^n \]

\[ \Delta; \Gamma \vdash A_3 [x_1 := M_1][x_2 := M_2] : s_3. \]

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 \( \Delta \vdash K_n =_\beta B[x_i := M_i]_{i=1}^n \). \( \Box \)

**Lemma 46**: Assume \( s \in S \) and \( M \) legal. If \( \Delta \vdash M =_\beta s \) then \( M \equiv s \).

**Proof:** First assume \( s \in \{\Box, \Delta\} \). If \( \Delta; \Gamma \vdash M : N \) for some \( \Gamma \) and \( N \), and \( \Delta \vdash M =_\beta s \) then by Church-Rosser \( \Delta \vdash M \rightarrow_\beta s \), so by Subject Reduction \( \Delta; \Gamma \vdash s : N \), contradicting the Generation Lemma. If \( \Delta; \Gamma \vdash N : M \) and \( \Delta \vdash M =_\beta s \) and \( s \neq s \) 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 \( s \equiv \ast \), \( \Delta; \Gamma \vdash M : N \), and \( \Delta \vdash M =_\beta s \). By Church-Rosser, \( \Delta \vdash M \rightarrow_\beta \ast \), say \( \Delta \vdash M \rightarrow_\beta \ast \rightarrow_\beta M' \rightarrow_\beta \ast \). By Subject Reduction, \( \Delta; \Gamma \vdash M' : N \) and \( \Delta; \Gamma \vdash \ast : N \). By Generation \( \Delta \vdash N =_\beta \Box \), so \( N \equiv \Box \).

33
• $M' \equiv (\lambda x. A B) C$ and $\ast \equiv B[x := C]$. By Generation $\exists B'$ where $\Delta \vdash B' [x := C] \equiv_{\beta\delta} \square$ (so $B' [x := C] \equiv \square$), $\Delta; \Gamma \vdash (\lambda x. A B')$ and $\Delta; \Gamma \vdash C : A$. $C \equiv \square$ contradicts $\Delta; \Gamma \vdash C : A$, so $B' \equiv \square$. By Lemma 32, $\Delta; \Gamma \vdash (\Pi x : A) \ast$, so by Generation $\Delta; \Gamma; x : A \vdash \square : \ast$, contradiction;

• $M' \equiv b M_1 \cdots M_n$ and $\Delta \vdash b M_1 \cdots M_n \rightarrow_\delta T[x_i := M_i]_{i=1}^n \equiv \ast$ as above.

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

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

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

- $|x|_\Delta \equiv x$ for $x \in \mathcal{V};$
- $|s|_\Delta \equiv s$ for $s \in \mathcal{S};$
- $|PQ|_\Delta \equiv |P|_\Delta |Q|_\Delta$
- $|\lambda x. P|_\Delta \equiv \lambda x. |P|_\Delta$
- $|\Pi x. P|_\Delta \equiv \Pi x. |P|_\Delta$
- $|\eta x. P|_\Delta \equiv \eta x. |P|_\Delta$
- $|b|_\Delta \equiv b$ for all $b \in \mathcal{C} \setminus \text{defcons}(\Delta;)$
- $|\lambda x_1 \ldots x_n. A|_\Delta \equiv |A|_\Delta[x_1, \ldots, x_n := T]|_\Delta$ if $\Delta \equiv \Delta_1, b = (x_1 := A_1, T), (x_2 := A_2, U), \Delta_2$

The following lemma is useful:

**Lemma 48** Let $\Delta_1, \Delta_2$ be left parts of legal contexts and $M, N \in \mathcal{T}$.

1. $\text{fv}(|M|_\Delta) = \text{fv}(M)$.
2. If $\Delta_2 \equiv \Delta_1, \Delta'$ and $\text{fc}(M) \subseteq \text{dom}(\Delta_1)$ 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 $\Delta \equiv \Delta_1, b := (\{T \vdash T\} : \{\Gamma_1 \vdash \Delta_2 \} : (T \neq \frac{1}{y}T_1, T_2))$. By the Definition Lemma, $T$ is typable in $\Delta_1, \Gamma$; therefore $\text{fv}(T) \subseteq \text{dom}(\Delta_1)$ (Free Variable Lemma). By IH, $\text{fv}(|T|_{\Delta_1}) \subseteq \text{dom}(\Delta_1)$ and therefore $\text{fv}(|b|_{\Delta_1}) = \emptyset$.
2. is by an easy induction on the definition of $|M|_\Delta$.
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 $\text{fv}(|M|_\Delta) = \text{fv}(M) = \emptyset$ (Lemma 48.1) and therefore $|M|_{\Delta} [x := |N|] \equiv |M[x := N]|_{\Delta}$.

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

**Lemma 49**
1. If $M \rightarrow_\beta N$ then $|M|_\Delta \rightarrow^+_\beta |N|_\Delta$.
2. If $\Delta \vdash M \rightarrow_\delta N$, then $|M|_\Delta \rightarrow^+_\beta |N|_\Delta$. 

34


\textbf{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 $|M|_\Delta \equiv (\lambda x:|P|_\Delta .|Q|_\Delta )|R|_\Delta \rightarrow_\beta |Q|_\Delta |R|_\Delta \equiv |Q[x:=R]|_\Delta$.

2. is by induction on the structure of $M$. We only treat the case in which $M \equiv b,M_1 \cdots M_n$; $\Delta \equiv \Delta_1$, $b = \left(\begin{array}{c} n \\ i \end{array}\right)x_i A_i T_i$; $(\prod_{i=1}^{n} x_i : A_i, U), \Delta_2$; and $N \equiv T[x_1, \ldots , x_n : M_1, \ldots , M_n]$. Note that $|M_i|_\Delta \equiv (\prod_{i=1}^{n} x_i : A_i \Delta_i, |T_i|_\Delta )|M_1|_\Delta \cdots |M_n|_\Delta \rightarrow_\beta |T_i|_\Delta |x_i : M_i|_\Delta \equiv (\prod_{i=1}^{n} x_i : M_i|_\Delta )|T_i|_\Delta \equiv |T[x_1, \ldots , x_n : M_1, \ldots , M_n]|_\Delta$.

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

Let $\textit{ASN}$ be the PTS over $\lambda$-terms with variables from $\forall \cup C$ and sorts from $S$, and the rules\footnote{We choose the name $\textit{ASN}$ because this system will help us in showing that $\lambda 68$ is SN.} $(*,*,*)$; $(*,*,\Delta)$; $(\rightarrow,*,\Delta)$; $(\rightarrow,\rightarrow,\Delta)$; $(\rightarrow,\Delta,\Delta)$; $(\rightarrow,\Delta,\Delta)$; $(\rightarrow,\rightarrow,\Delta)$.

This is in fact the pure type system that is based on the II-formation rules of Section 3.1. $\textit{ASN}$ is contained in $\textit{ECC}$ [2]. As $\textit{ECC}$ is $\beta$-strongly normalising, also $\textit{ASN}$ is $\beta$-strongly normalising.

We present a translation of $\lambda 68$-contexts to $\lambda$-contexts:

\textbf{Definition 50} Let $\Delta; \Gamma$ be a legal $\lambda 68$-context.

- We define $|\Delta|$ by induction on the length of $\Delta$:
  
  $|\emptyset| \ \overset{\text{def}}{=} \emptyset$; \hspace{1cm} $|\Delta, b : U|_\Delta \overset{\text{def}}{=} |\Delta|, b : |U|_\Delta$; \hspace{1cm} $|\Delta, b := T : U| \overset{\text{def}}{=} |\Delta|$

- 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 $\Delta$ are not translated into $|\Delta|$. This corresponds to the fact that all these definitions are unfolded (replaced by their definition) in $|b|_\Delta$. Now we prove a very important lemma:

\textbf{Lemma 51} If $\Delta; \Gamma \vdash_{\lambda 68} M : N$ then $|\Delta; \Gamma| \vdash_{\textit{ASN}} |M|_\Delta : |N|_\Delta$.

\textbf{Proof: } By induction on the derivation of $\Delta; \Gamma \vdash M : N$. We treat the cases:

\textbf{(Start: Primitive Constants)} $\Delta; \Gamma \vdash_{\lambda 68} B : s_1$; $\Delta; \vdash_{\textit{ASN}} |\Gamma, B|_\Delta$; $s_2$; $s_1 = *, \square$

By IH, $|\Delta| \vdash_{\textit{ASN}} |\Gamma, B|_\Delta$; $s_2$, so by Start $|\Delta|, b ; |\Gamma, B|_\Delta \vdash b ; |\Gamma, B|_\Delta$.

Observe that $|\Delta|, b ; |\Gamma, B|_\Delta \equiv |\Delta|, b ; |\Gamma, B|_\Delta$; that $|b|_\Delta \equiv |\Gamma, B|_\Delta$ and that (by Lemma 48.2) $|\Gamma, B|_\Delta \equiv |\Gamma, B|_\Delta, b ; |\Gamma, B|_\Delta$.

\textbf{(Start: Defined Constants)} $\Delta; \Gamma \vdash_{\lambda 68} T : B : s_1$; $\Delta; \vdash_{\textit{ASN}} |\Gamma, B|_\Delta$; $s_2$; $s_1 = *, \square$

By induction $|\Delta| \vdash_{\textit{ASN}} |\Gamma, B|_\Delta$; $s_2$, so (write $\Gamma \equiv x_1 : A_1, \ldots , x_n : A_n$):

$$|\Delta| \vdash_{\textit{ASN}} \prod_{i=1}^{n} x_i : |A_i|_\Delta, \prod_{i=1}^{n} x_i : |B|_\Delta : s_2.$$
By induction, we also have $|\Delta; \Gamma| \vdash_{\text{ASN}} |T|_{\Delta} : |B|_{\Delta}$, so:
\[
|\Delta|, x_i: |A_i|_{\Delta}, \ldots, x_n: |A_n|_{\Delta} \vdash_{\text{ASN}} |T|_{\Delta} : |B|_{\Delta},
\]
and by repeatedly applying the $\lambda$-rule on (13) and using the fact that, by IH, the types $\prod_{i=1}^{n} x_j: |A_j|_{\Delta}$ are all typable, we find:
\[
|\Delta| \vdash_{\text{ASN}} \left( \prod_{i=1}^{n} x_i: |A_i|_{\Delta} \cdot |T|_{\Delta} \right) : \left( \prod_{i=1}^{n} x_i: |A_i|_{\Delta} \cdot |B|_{\Delta} \right);
\]

(Application 1) \[\frac{\Delta; \Gamma \vdash_{\text{M68}} M : (\Pi x: A. B)}{\Delta; \Gamma \vdash_{\text{M68}} MN : B[x:=N]}\]

By IH, $|\Delta; \Gamma| \vdash_{\text{ASN}} |M|_{\Delta} : (\Pi x: |A|_{\Delta}. |B|_{\Delta})$ and $|\Delta; \Gamma| \vdash_{\text{ASN}} |N|_{\Delta} : |A|_{\Delta}$.

By application $|\Delta; \Gamma| \vdash_{\text{ASN}} |M|_{\Delta} |N|_{\Delta} : |B|_{\Delta} [x:=|A|_{\Delta}]$. By definition of $|MN|_{\Delta}$ and Lemma 48.3, $|\Delta; \Gamma| \vdash_{\text{ASN}} |MN|_{\Delta} : |B[x:=A]|_{\Delta}$.

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

**Proof:** Assume, we have an infinite $\beta\delta$-reduction path in $\lambda\delta$:
\[
M_1 \rightarrow_{\beta\delta} M_2 \rightarrow_{\beta\delta} M_3 \rightarrow_{\beta\delta} \ldots
\]

As $\delta$-reduction is strongly normalising (Lemmas 20 and 35), there must be infinitely many $\beta$-reductions in this reduction path, so we have a path $N_1 \rightarrow_{\beta} N_2 \rightarrow_{\beta} N_3 \rightarrow_{\beta} N_4 \rightarrow_{\beta} \ldots$ By Lemmas 49.1 and 49.2, this gives a path $|N_1|_{\Delta} \rightarrow_{\beta} |N_2|_{\Delta} \rightarrow_{\beta} |N_3|_{\Delta} \rightarrow_{\beta} |N_4|_{\Delta} \rightarrow_{\beta} \ldots$ which is an infinite $\beta$-reduction path in $\lambda\text{SN}$. By Lemma 51, $|N_1|_{\Delta}$ is legal in $\lambda\text{SN}$. But as $\lambda\text{SN}$ is strongly normalising, this infinite $\beta$-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 $\text{AUT-68}$ and $\lambda\delta$.

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

- If $\mathcal{B}; \Gamma \vdash_{\text{AUT-68}} \text{OK}$ then $\mathcal{B}; \Gamma$ is legal;
- If $\mathcal{B}; \Gamma \vdash_{\text{AUT-68}} \Sigma : \Omega$ then $\mathcal{B}; \Gamma \vdash_{\text{M68}} \Sigma : \Omega$.

**Proof:** We prove both statements simultaneously, by induction on the derivation of $\mathcal{B}; \Gamma \vdash_{\text{AUT-68}} \text{OK}$ and $\mathcal{B}; \Gamma \vdash_{\text{AUT-68}} \Sigma : \Omega$ of Definitions 14 and 15. We only treat one case. Assume, the last step is book extension rule def2:
\[
\mathcal{B}; \Gamma \vdash_{\text{AUT-68}} \Sigma_2 : \text{type} \quad \mathcal{B}; \Gamma \vdash_{\text{AUT-68}} \Sigma_1 : \Sigma_2 \quad \mathcal{B}; \Gamma \vdash_{\text{AUT-68}} \Sigma_2 =_{\beta\delta} \Sigma'_2.
\]

By IH, we have
\[
\mathcal{B}; \Gamma \vdash_{\text{M68}} \Sigma_2 : *
\]
and
\[
\mathcal{B}; \Gamma \vdash_{\text{M68}} \Sigma_1 : \Sigma_2.
\]
By Lemma 35, we have
\[ \overline{B}; \Gamma \vdash_{\lambda 68} \Sigma_2 =_{\beta \lambda} \Sigma_2. \] (18)

Applying the conversion rule of \( \lambda 68 \) to (16), (17) and (18) yields
\[ \overline{B}; \Gamma \vdash_{\lambda 68} \Sigma_1 : \Sigma_2. \] (19)

As \( \overline{B}; \Gamma \) is legal, for each \( x \alpha \in \Gamma \) (say: \( \Gamma \equiv \Gamma_1, x \alpha, \Gamma_2 \)) we have \( \overline{B}; \Gamma_1 \vdash \alpha : s \) for an \( s \in \{*, \Box \} \), by the Free Variable Lemma 26. Thus we can repeatedly apply the \( \Box \)-formation rule (starting with (16)) to obtain:
\[ \overline{B}; \Gamma \vdash_{\lambda 68} \Box \Gamma \Sigma_2 : \Delta. \] (20)

(If \( \Gamma \equiv \emptyset \) then we apply the \( \Box \)-formation rule zero times, and the type of \( \Box \Gamma \Sigma_2 \) is \( * \) instead of \( \Delta \)). Now we can apply the (Start: dc) rule on (19), (16) and (20) to obtain: \( \overline{B}; k := (\Box \Gamma \Sigma_1) : (\Box \Gamma \Sigma_2) : \vdash_{\lambda 68} \overline{B}; k : (\Box \Gamma \Sigma_2) \) is legal.

**Theorem 54** Let \( \Delta; \Gamma \vdash_{\lambda 68} M : N \). There is an AUTOMATH book \( \mathcal{B} \) and context \( \Gamma' \) such that \( \mathcal{B}; \Gamma' \vdash_{\text{AUT-68}} \Omega \), and \( \overline{B}; \Gamma' \equiv \Delta; \Gamma \). Moreover,
1. If \( \Gamma \equiv \emptyset \) then \( M \equiv * \);
2. If \( \Delta; \Gamma \vdash_{\lambda 68} N : \Box \) then \( N \equiv * \) and there is \( \Omega \in \mathcal{E} \) such that \( \overline{B}; \Omega \equiv M \) and \( \overline{B}; \Gamma' \vdash_{\text{AUT-68}} \Omega : \text{type} \);
3. If \( \Gamma \equiv \Delta \) then there is \( \Gamma'' \equiv x_1 : \Sigma_1, \ldots, x_n : \Sigma_n \) and \( \Omega \in \mathcal{E} \cup \{\text{type}\} \) such that: \( \bullet \Gamma', \Gamma'' \) is correct with respect to \( \mathcal{B} \); \( \bullet \Omega \equiv \text{type} \) or \( \overline{B}; \Gamma' \vdash_{\text{AUT-68}} \Omega : \text{type} \);
4. If \( \Delta; \Gamma \vdash_{\lambda 68} N : \Delta \) then there are \( b \in \mathcal{C} \) and \( \Sigma_1, \ldots, \Sigma_n \in \mathcal{E} \) such that \( M \equiv b \Sigma_1 \cdot \cdots \cdot \Sigma_n \). Moreover, \( \mathcal{B} \) contains a line \( \{x_1 : \Omega_1, \ldots, x_m : \Omega_m, b : \Sigma_1, \Sigma_2\} \) such that: \( \bullet \Gamma \equiv \{b \Sigma_1, \ldots, \Sigma_n\} \); \( \bullet M \equiv b \Sigma_1 \cdot \cdots \cdot \Sigma_n \).
5. If \( \Gamma \equiv * \) then \( \exists \Omega \in \mathcal{E} \) such that \( \overline{B}; \Omega \equiv M \) and \( \overline{B}; \Gamma' \vdash_{\text{AUT-68}} \Omega : \text{type} \);
6. If \( \Delta; \Gamma \vdash_{\lambda 68} N : * \) then there are \( \Sigma_1, \Sigma_2 \in \mathcal{E} \) such that \( \Sigma \equiv M \) and \( \Omega \equiv N \), and \( \overline{B}; \Gamma' \vdash_{\text{AUT-68}} \Sigma_1 : \Sigma_2 \equiv \Omega \equiv N \), and \( \overline{B}; \Gamma' \vdash_{\text{AUT-68}} \Sigma_1 : \text{type} \).

**Proof:** Induction on the derivation of \( \Delta; \Gamma \vdash_{\lambda 68} M : N \). We treat the cases:

**Weakening definitions.** Assume the last step is
\[ \Delta; \Gamma \vdash_{\lambda 68} M : N \quad \Delta; \Gamma \vdash_{\lambda 68} T : B \quad \Delta; \Gamma \vdash_{\lambda 68} \Box \Gamma : s_2 \] where \( s \equiv * \) or \( s \equiv \Box \). Use IH and determine \( \mathcal{B}, \Gamma', \Sigma_1, \Sigma_2, \Omega_1, \) and \( \Omega_2 \) such that \( \overline{B} \equiv \Delta, \overline{B} \equiv \Gamma, \Sigma_1 \equiv T, \Sigma_2 \equiv B, \Omega_1 \equiv M \) and \( \Omega_2 \equiv N \). We know by induction that \( \mathcal{B}, \Gamma' \vdash_{\text{AUT-68}} \Sigma_2 : \text{type} \) (if \( s \equiv * \)) or \( \Sigma_2 \equiv * \) (if \( s \equiv \Box \)). Also \( \mathcal{B}, \Gamma' \vdash_{\text{AUT-68}} \Sigma_1 : \text{type} \). This makes it possible to extend \( \mathcal{B} \) with a new line, thus obtaining a legal book \( \mathcal{B}, (\Gamma'; b, \Sigma_1, \Sigma_2) \). Using Weakening for \( \text{AUT-68} \) (Lemma 22) and IH on \( \Delta; \Gamma \vdash_{\lambda 68} M : N \), it is not hard to verify the cases 1–6 for \( \Delta, b := (\Box \Gamma \Box) : (\Box \Gamma \Box) : \vdash_{\lambda 68} M : N \);
Application 2 The last step is
\[
\Delta; \Gamma \vdash_{\lambda 68} M_1 : (\xi(x:A).B) \quad \Delta; \Gamma \vdash_{\lambda 68} M_2 : A.
\]

Determine \( \mathcal{B}, \Gamma' \) such that \( \mathcal{B} \equiv \Delta \) and \( \Gamma' \equiv \Gamma \). By Correctness of Types 32 and Generation Lemma 31, \( \Delta; \Gamma \vdash_{\lambda 68} (\xi(x:A).B) : \Delta \), so by IH (case 4), there are \( b, \Sigma_1, \ldots, \Sigma_n \) such that \( M_1 \equiv b \Sigma_1 \cdots \Sigma_n \), and there is a line \( (x_1; \Omega_1, \ldots, x_m; \Omega_m; b; \xi; \Omega; \Sigma_1; \xi; \Omega[x_j;=\xi]_{j=1}^{n} \models 1 \leq i \leq n \) and \( \xi(x:A).B \equiv (\xi^{m}_{i=1} x_i; \Omega_i; \Sigma_j) [x_j;=\xi]_{j=1}^{n+1} \).

Observe: \( A \equiv \Omega_{n+1}[x_j;=\xi]_{j=1}^{n+1} \). As \( \mathcal{B}, \Gamma' \vdash_{\lambda 68} \Omega_{n+1}[x_j;=\xi]_{j=1}^{n+1} : \text{type} \) for an \( s \in \{*, \Box\} \), and by Substitution and Transitivity Lemmas we have \( \Delta; \Gamma \vdash_{\lambda 68} \Omega_{n+1}[x_j;=\xi]_{j=1}^{n+1} : \text{type} \). With IH we determine \( \Sigma \in \mathcal{E} \) such that \( \mathcal{B}, \Gamma' \vdash_{\lambda 68} \Sigma : \Omega_{n+1}[x_j;=\xi]_{j=1}^{n+1} \) and \( M_2 \equiv \Sigma \).

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 \( B[x;=M_2] \equiv \xi_{2}[x;=\xi]_{j=1}^{n+1} \). As \( \Delta; \Gamma \vdash_{\lambda 68} B[x;=M_2] : \Delta \), \( \xi_{2}[x;=\xi]_{j=1}^{n+1} \) is of the form \( \xi(x:P,Q) \), which is impossible.

6. Note: \( B[x;=M_2] \equiv (\xi_{2}^{m}_{i=1} x_i; \Omega_i; \Sigma_i) [x_i;=\xi]_{j=1}^{n+1} \). We have \( \Delta; \Gamma \vdash_{\lambda 68} B[x;=M_2] : * \). So \( B[x;=M_2] \neq \xi_{2}; P; Q \), and hence \( m = n + 1 \). Therefore, \( \mathcal{B}, \Gamma' \vdash_{\lambda 68} \Omega_{n+1}[x_j;=\xi]_{j=1}^{n+1} : \Sigma \) by \( \text{type} \).

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

- The cases \( \Sigma \equiv \Box \) 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 \( \text{type} \) is the only "top-expression" in \( \text{AUTO-68} \).

- The cases \( \Sigma \equiv * \) and \( \Delta; \Gamma \vdash N : * \) give a precise correspondence between expressions of \( \text{AUTO-68} \) and terms of \( \lambda 68 \): If \( M : N \) in \( \lambda 68 \) then there are \( \Sigma, \Omega \) in \( \text{AUTO-68} \) such that \( \Sigma : \Omega \) in \( \lambda 68 \) and \( \Sigma \equiv M \) and \( \Omega \equiv N \).

- The cases \( \Sigma \equiv \Delta \) and \( \Delta; \Gamma \vdash N : \Delta \) cover terms that do not have an equivalent in \( \text{AUTO-68} \) but are necessary in \( \lambda 68 \) to form terms that have equivalents in \( \text{AUTO-68} \). More specific, this concerns terms of the form \( \xi_{i=1}^{m} x_i ; A_i \cdot B \) (needed to introduce constants) and terms of the form \( b M_1 \cdots M_n \), where \( b \) is of constant of type \( \xi_{i=1}^{m} x_i ; A_i \cdot B \) 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 \( \text{AUTO-68} \) coincide as much as possible, and that the terms in \( \lambda 68 \) that do not have an equivalent in \( \text{AUTO-68} \) can be traced easily (these are the terms of type \( \Delta \) and the terms of a type \( N : \Delta \), and the sorts \( \Box \) and \( \Delta \), which are needed to a give a type to \( * \) and to the \( \xi \)-types).

Notice that the alternative definition of \( \delta \)-reduction in \( \lambda 68 \), discussed at the end of Subsection 2.5, would introduce more terms in \( \lambda 68 \) without an equivalent in \( \text{AUTO-68} \), namely terms of the form \( \lambda_{i=1}^{m} x_i ; A_i \cdot B \).
5 More suitable PTSs for \textsc{Automath} systems

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

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

nor $\Pi$-application where the $\lambda 68$ rule $(\text{App}_x)$ is changed into

\[ \Pi\text{-application} \quad \frac{\Delta; \Gamma \vdash M : \Pi x : A. B \quad \Delta; \Gamma \vdash N : A}{\Delta; \Gamma \vdash M N : (\Pi x : A. B) N} \]

There are good reasons to use parameters (cf. \cite{32, 33}), $\Pi$-reduction and $\Pi$-application (cf. \cite{31, 36}). In Section 5.1 we look at how we might remedy the above shortcomings to create more faithful interpretations of \textsc{Aut-68} as PTSs.

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

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

PTSs don’t usually follow \textsc{Automath} in identifying $\text{ls}$ and $\text{IIs}$. PTSs don’t even follow \textsc{Automath} in allowing $\Pi$-reduction and $\Pi$-application. We have the following results in the area:

- \cite{30} showed that as long as the usual application rule of PTSs is used, a PTS system remains unchanged whether $\Pi$-reduction is included or not. As a result, if the usual application rule of PTSs is used, a PTS system remains unchanged whether $\text{ls}$ and $\text{IIs}$ are unified or not. \cite{30} concluded that a PTS system where $\text{ls}$ and $\text{IIs}$ are unified and where the application is changed to $\Pi$-application faces the same problem (and inherits the same solution) as that of the PTSs where $\text{ls}$ and $\text{IIs}$ are not unified but where $\Pi$-application and $\Pi$-reduction are used.

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

- \cite{31} showed that PTSs with $\Pi$-reduction and $\Pi$-application have all the desirable properties if a definition system is used. Let us call the PTS with $\Pi$-reduction and $\Pi$-application and definitions as in \cite{31}, $\lambda \beta \Pi \text{Id}$. 

39
Though our system $\lambda 68$ does not have $\Pi$-reduction and $\Pi$-application, it is easy to extend it to a system $\lambda 68 \Pi$ by adding these rules:

- Changing rule $(\text{App}_1)$ into
  \[
  \frac{\Delta; \Gamma \vdash M : \Pi x. A.B \quad \Delta; \Gamma \vdash N : A}{\Delta; \Gamma \vdash MN : (\Pi x. A.B)N}
  \]
  (Rule $(\text{App}_2)$ remains unchanged—see also the discussion in Section 3.1);

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

The system $\lambda 68 \Pi$ is actually much closer to $\text{Aut-68}$ than $\lambda 68$.

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

\[ ;\alpha; \ast, x; \alpha \vdash (\lambda y. \alpha.y)x : (\Pi y. \alpha.\alpha)x \]

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

\[ ;\alpha; \ast, x; \alpha \vdash x : (\Pi y. \alpha.\alpha)x \]

The “restoration” of Subject Reduction in $\lambda \Pi \Pi$ is only because of the special way in which definitions are introduced and removed from the context. In $\lambda 68 \Pi$, 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 $\lambda 68 \Pi$ in order to restore Subject Reduction in $\lambda 68 \Pi$.

As for parameters, [32] gives a formulation of PTSs with parameters, [33] formulates PTSs with parameters, $\Pi$-reduction, $\Pi$-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 $\lambda$s and $\Pi$s are unified, and with parameters, $\Pi$-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 $C(A_1, \ldots, A_n)$ where $C$ is a set of constants disjoint from the set of variables, and $n \geq 0$. Then, in addition to the set of $(\Pi$-formation) rules $R$, a set of parametric construction rules $P$ is added. Typing rules for dealing with the new terms are finally added as follows: $(\Delta \equiv x_1:B_1, \ldots, x_n:B_n, \Delta_i \equiv x_i:B_1, \ldots, x_{i-1}, x_{i+1}:B_{i-1} \text{ and } \text{CONS } (\Gamma))$:

\[
(\bar{C}\text{-weak}) \quad \frac{\Gamma \vdash_a C : B \quad \Gamma, \Delta; \Gamma_i \vdash_s B_i : s_i \quad \Gamma, \Delta, A : s \vdash_a (s_i, s) \in P, \ c \not\in \text{CONS } (\Gamma)}{\Gamma, c(\Delta) : A \vdash_a C : B}
\]

\[
(\bar{C}\text{-app}) \quad \frac{\Gamma, c(\Delta) : A \vdash_a A_1 : A_1 \vdash A_i[B_i[x_j:=A_j]_{j=1}^{i-1}] \quad (i = 1, \ldots, n)}{\Gamma, c(\Delta) : A \vdash_a A : s}
\]

\[ \text{if } n = 0 \]

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 \([x: \Sigma_1] \text{type}\) (extending Definition 8);

2. Inhabitants of types of the form \([x: \Sigma_1] \text{type}\) are introduced by extending the abstraction rules 1 and 2 of Definition 15 with the rule for Aut-QE:

\[
\begin{align*}
\forall \Theta; \Gamma & \vdash \Sigma_1: \text{type} \\
\forall \Theta; \Gamma, x: \Sigma_1 & \vdash \Sigma_2: \text{type} \\
\forall \Theta; \Gamma & \vdash [x: \Sigma_1][\Sigma_2 : [x: \Sigma_1] \text{type}]
\end{align*}
\]

Notice that the expression \([x: \Sigma_1] \text{type}\) is not typable, just as \(\text{type}\) is not typable. In a translation to a PTS, these expressions should get type \(\square\);

3. There is a new reduction relation on expressions, which is specific for Aut-QE (which we call \(\rightarrow_{\text{QE}}\) in the sequel). This relation is given by the rule \([x_1: \Sigma_1] \cdots [x_n: \Sigma_n][\Omega] : \text{type} \rightarrow_{\text{QE}} [x_1: \Sigma_1] \cdots [x_n: \Sigma_n] : \text{type}\) (for \(n \geq 0\)).

The first two rules are rather straightforward. They correspond to an extension of \(\lambda\to\) to \(\lambda P\) in Pure Type Systems. It is also easy to extend \(\lambda 68\) with similar rules: We just add the \(\Pi\)-formation rule \((*, \Box, \Box)\):

\[
\begin{align*}
\Delta, \Gamma & \vdash A : * \\
\Delta & \vdash \Pi x : A : B : \Box \\
\Delta, \Gamma & \vdash (\Pi x : A : B) : \Box
\end{align*}
\]

In Aut-68, \(\Pi A\) 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 \([x: \Sigma] \text{type}\), 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 \(\lambda 8\) and \(\Pi 8\). 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 $\text{type}$, and the first should not have type $\text{type}$. In $\text{AUT-QE}$ the situation is more complicated. An expression $[x;\Sigma]\Omega$ may have more than one type:

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

$$
(\varnothing, \alpha, -, \text{type}),
(\alpha, \text{type}, x, -, \alpha).
$$

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

$$
\mathcal{B}; \alpha, \text{type} \vdash_{\text{QE}} [x;\alpha] \alpha : \text{type}.
$$

But using the new abstraction rule of $\text{AUT-QE}$ we can also derive

$$
\mathcal{B}; \alpha, \text{type} \vdash_{\text{QE}} [x;\alpha] \alpha : [x\alpha] \text{type}.
$$

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

$$
\mathcal{B}; \Gamma \vdash_{\text{QE}} [x_{1};\Sigma_{1}] \cdots [x_{n};\Sigma_{n}] \Omega : [x_{1};\Sigma_{1}] \cdots [x_{n};\Sigma_{n}] \text{type},
$$

$$
\mathcal{B}; \Gamma \vdash_{\text{QE}} [x_{1};\Sigma_{1}] \cdots [x_{m};\Sigma_{m}] \Omega : [x_{1};\Sigma_{1}] \cdots [x_{m};\Sigma_{m}] \text{type}
$$

(for $m < n$). In (23), the expression $[x_{1};\Sigma_{1}] \cdots [x_{n};\Sigma_{n}] \Omega$ should be read as $\lambda_{n}^{m} x_{i};\Sigma_{i} \Omega$; in (24) it should be read as $\lambda_{m}^{n} x_{j};\Sigma_{j} \Omega$.

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 $\Sigma$ (take, for instance, $\Sigma$ a variable). In order that the equivalence holds for general expressions $\Sigma$, de Bruijn introduced a rule for type inclusion:

$$
\mathcal{B}; \Gamma \vdash_{\text{QE}} \Sigma : [x_{1};\Sigma_{1}] \cdots [x_{n};\Sigma_{n}] \text{type} \quad \overline{\mathcal{B}; \Gamma \vdash_{\text{QE}} \Sigma : [x_{1};\Sigma_{1}] \cdots [x_{n+1};\Sigma_{n+1}] \text{type}}.
$$

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 $\lambda$-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 $\langle B \rangle [x ; s] [y ; x] y$ instead of $(\lambda x : s, \lambda y : x)y$.

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

In de Bruijn’s notation, the $\beta$-rule $(\lambda x : C.A)B \rightarrow_{\beta} A[x := B]$ becomes:

$$
\langle B \rangle [x ; C]A \rightarrow_{\beta} [x := B]A
$$

42
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 \( \beta \)-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):

<table>
<thead>
<tr>
<th>Classical Notation</th>
<th>De Bruijn’s Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>((\lambda x. (\lambda y. \lambda z. zD) \downarrow^+ B) \downarrow^o A)</td>
<td>((\lambda A. \langle B \rangle[x] (\langle C \rangle[y][z] \langle D \rangle z) \downarrow \beta)</td>
</tr>
<tr>
<td>((\lambda y. \lambda z. zD) \downarrow^+ C \downarrow \beta)</td>
<td>((\lambda A \langle C \rangle[y][z] \langle D \rangle z) \downarrow \beta)</td>
</tr>
<tr>
<td>((\lambda x. \lambda z. zD) \downarrow \beta)</td>
<td>((\langle A \rangle[z] \langle D \rangle z) \downarrow \beta)</td>
</tr>
<tr>
<td>(AD)</td>
<td>((\langle D \rangle A))</td>
</tr>
</tbody>
</table>

The bracketing structure in classical notation of \((\lambda x. (\lambda y. \lambda z. zD) \downarrow^+ B) \downarrow^o A\), is \([1 \uparrow^+ 2 \uparrow 3]\), where \([1\] and \([2\] match. Whereas \langle A \rangle \langle B \rangle (x) \langle C \rangle \langle y \rangle \langle z \rangle \langle D \rangle z has the simpler bracketing structure \([1][2][3]\) or even better: \([1][2][3]\) 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 \(\beta\)-reduction, which keeps the AT-pair and does \(\beta\)-reduction at one instance (instead of all the instances). For example (we take a simpler example than above and again ignore types for simplicity):

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

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

\(\langle B \rangle \langle x \rangle \langle C \rangle \langle A \rangle \sim^\beta \langle A \rangle \langle x \rangle \langle C \rangle \langle D \rangle \langle z \rangle\) where \(\langle x \rangle\) is a well balanced segment.

So for example, \(\langle A \rangle \langle B \rangle \langle x \rangle \langle C \rangle \langle y \rangle \langle z \rangle \langle D \rangle z \sim^\beta \langle B \rangle \langle x \rangle \langle C \rangle \langle y \rangle \langle z \rangle \langle D \rangle z\).

The \(\lambda\)-calculus à la de Bruijn has many advantages over the classical \(\lambda\)-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 developed 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)\ref{footnote:lambdaDeltaLambda} but where $\beta$-reduction\ref{footnote:betaReduction} is presented as the result of local $\beta$-reductions and AT-removals. The reason for this is that the delta reductions of AUTOMATH can be considered as local $\beta$-reductions, and not as ordinary $\beta$-reductions.

We have fully investigated PTSs and the type free lambda calculus in de Bruijn's notation \cite{deBruijn58,deBruijn68}. We have also shown that $\sim_\beta$ satisfies nice properties in the type free lambda calculus \cite{Girard87} and that it loses subject reduction in PTSs but that subject reduction can be regained if definitions are added in the contexts \cite{Barendregt84}. We have not yet studied PTSs with local $\beta$-reductions and AT-removal, although we have studied the type free lambda calculus with local $\beta$-reduction, AT-removal and explicit substitution \cite{Barendregt92}. We leave the study of PTSs with de Bruijn's local $\beta$-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 \cite{Barendregt84,Barendregt86}, we feel that our description is more accurate. Moreover, unlike \cite{Barendregt84,Barendregt86}, 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 $\Pi$-conversion (while AUTOMATH does), one can adapt it to include $\Pi$-conversion following the lines of \cite{Barendregt84a}.

The adaptation of $\lambda 68$ to a system $\lambda$QE, representing the AUTOMATH-system AUT-QE is not hard, either: it requires adaptation of the $\Pi$-formation rule to include not only the rule $(\ast,\ast,\ast)$ but also $(\ast,\Box,\Box)$ 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 $\beta$-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 $\lambda$-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

\begin{enumerate}
\end{enumerate}

\footnote{In $\Delta\Lambda$, de Bruijn favours trees over character strings and does not make use of AT-couples.}

\footnote{Recall this is now both $\beta$ and $\Pi$-reduction as he unifies $\Lambda$s and $\Pi$s.}


