URL: http://www.elsevier.nl/locate/entcs/volume85.html 20 pages

Automath and Pure Type Systems

Fairouz Kamareddine [1]

School of Mathematical and Computational Sciences
Heriot-Watt Univ., Riccarton
Edinburgh EH14 4AS, Scotland

Twan Laan [2]

Weerdstede 45, 3431 LS Nieuwegein, The Netherlands

Rob Nederpelt [3]

Mathematics and Computing Science,
Eindhoven Univ. of Technology,
P.O.Box 513, 5600 MB Eindhoven, the Netherlands

Abstract

We study the position of the AUTOMATH systems within the framework of Pure Type Systems (PTSs). In [1,15], 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, 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 [13] and Nuprl [10]. 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." ([8]; see [24] p. 201)

[1] Email: fairouz@macs.hw.ac.uk
[2] Email: twan.laan@wxs.nl
[3] Email: r.p.nederpelt@tue.nl

Thus, the roots of Automath are not to be found in logic or type theory, but in mathematics and the mathematical vernacular [7]. For some years, de Bruijn had been wondering what a proof of a theorem in mathematics should be like, and how its correctness can be checked. The development of computers in the sixties made him wonder whether a machine could check the proof of a mathematical theorem, provided the proof is written in a very accurate way. De Bruijn developed the language Automath for this purpose. This language is not only (according to de Bruijn [6]) "*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*". The goals of the Automath project were given as:

"1. The system should be able to verify entire mathematical theories.
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." ([8]; see [24] pp. 209–210)

Goal 1 was achieved: Van Benthem Jutting [2] translated and verified Landau's "Grundlagen der Analysis" [23] in Automath and Zucker [29] formalised classical real analysis in Automath.

As for goal 2, de Bruijn used types and a propositions as types (pat) principle[4] that was somewhat different from Curry and Howard's [11,17].

De Bruijn spent a lot of effort on goal 3 and studied the language of mathematics in depth [7]. Automath features that helped him in goal 3 include:

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

As Automath was developed independently from other developments in the world of type theory and λ-calculus, and as it invented powerful typing ideas that were later adopted in influential type systems (cf. [1]), 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 "Begriffschrift" [14]. It was presented in 1910 in the famous "Principia Mathematica" [28] 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" [9]. In 1988-1989, Berardi [4] and Terlouw [27] gave as an extension of Barendregt's work [1], a general framework for type systems, which is at the basis of the so-

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

called Pure Type Systems (PTSs [1]). PTSs include many of the type systems that play an important role in programming languages and theorem proving.

In this paper we focus on the relation between Automath and Pure Type Systems (PTSs). Both [1] and [15] 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 [1] and [15] consider Automath without one of its most important mechanisms: definitions and parameters. But definitions and parameters are 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 (cf. Section 5 of [6]). According to de Bruijn [8] this is "*due to the fact that mathematicians worked with abbreviations all the time already*". Moreover, recent developments on the use of definitions and parameters in Pure Type Systems [18,26,19,20] justify renewed research on the relation between Automath and PTSs.

- In Section 2 we give a description of Aut-68, a basic Automath system.
- In Section 3 we discuss how we can transform Aut-68 into a PTS. In doing so, we notice that Aut-68 has some properties that are not usual for PTSs: • Aut-68 has η-reduction; • Aut-68 has Π-application and Π-reduction (as it does not distinguish λ and Π); • Aut-68 has a definition system; • Aut-68 has a parameter mechanism. We do not consider η-reduction as an essential feature of Automath, and focus on the definition and parameter mechanisms, which are the most characteristic type-theoretical features of Automath. In systems with Π-application, Π behaves like λ, and there is a rule of Π-reduction: $(\Pi x{:}A.B)N \to_\Pi B[x{:=}N]$. In Automath, both $\Pi x{:}A.B$ and $\lambda x{:}A.B$ are denoted by $[x{:}A]B$. It is not easy to see whether $[x{:}A]B$ represents $\lambda x{:}A.B$ or $\Pi x{:}A.B$. Fortunately, this is not a problem for Aut-68.
- In Section 4, we present a system $\lambda 68$ that is (almost) a PTS. We show that it has the usual properties of PTSs and we prove that $\lambda 68$ can be seen as Aut-68 without η-reduction, Π-application and Π-reduction.

2 Description of Automath

During the Automath-project, several Automath-languages were developed. They all have two mechanisms for describing mathematics. The first is essentially a typed λ-calculus, with the important features of λ-abstraction, λ-application and β-reduction. The second mechanism is the use of definitions and parameters. The latter is the same for most Automath-systems, and the difference between the various systems is mainly caused by the λ-calculi used. In this section we will describe the system Aut-68 [3,5,12] 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. We start with a review of PTSs.

2.1 Pure Type Systems

Definition 2.1 Let $\mathbb{V}$ be a set of variables and $\mathbb{C}$ a set of constants (both countably infinite). The set $\mathbb{T}(\mathbb{V}, \mathbb{C})$ (or $\mathbb{T}$, if it is clear which sets $\mathbb{V}$ and $\mathbb{C}$ are used) of typed lambda terms with variables from $\mathbb{V}$ and constants from $\mathbb{C}$ is defined by the following abstract syntax: $\mathbb{T} ::= \mathbb{V} \mid \mathbb{C} \mid \mathbb{T}\mathbb{T} \mid \lambda\mathbb{V}{:}\mathbb{T}.\mathbb{T} \mid \Pi\mathbb{V}{:}\mathbb{T}.\mathbb{T}$.

We use x, y, z, α, β as meta-variables over $\mathbb{V}$. In examples, we sometimes want to use some specific elements of $\mathbb{V}$; we use typewriter-style to denote such specific elements. So: $\mathtt{x}$ is a specific element of $\mathbb{V}$; while x is a meta-variable *over* $\mathbb{V}$. The variables $\mathtt{x}$, $\mathtt{y}$, $\mathtt{z}$ are assumed to be *distinct* elements of $\mathbb{V}$ (so $\mathtt{x} \not\equiv \mathtt{y}$ etc.), while meta-variables $x, y, z, \ldots$ may refer to variables in the object language that are syntactically equal. We use $A, B, C, \ldots, a, b, \ldots$ as meta-variables over $\mathbb{T}$. $\textsc{fv}(A)$, the set of *free variables* of A, and substitution $A[x{:=}B]$ are defined in the usual way. We use $\equiv$ to denote syntactical equality between typed lambda terms. Terms that are equal up to a change of bound variables are considered to be syntactically equal. We assume the *Barendregt Convention* [1] where bound variables are chosen to differ from free ones.

Note 1 • *We write $AB_1 \cdots B_n$ as shorthand for $(\cdots((AB_1)B_2)\cdots B_n)$.*

- *We write $\pi\boldsymbol{x}{:}\boldsymbol{A}.B$, or $\pi_{i=1}^{n} x_i{:}A_i.A$, as shorthand for $\pi x_1{:}A_1.(\pi x_2{:}A_2.(\cdots(\pi x_n{:}A_n.A)\cdots))$; for $\pi \in \{\lambda, \Pi\}$*
- *We use the abbreviation $A[x_i{:=}B_i]_{i=m}^{n}$ to denote $A[x_m{:=}B_m]\cdots[x_n{:=}B_n]$. If $m > n$ then $A[x_i{:=}B_i]_{i=m}^{n}$ denotes A. We write $A[\boldsymbol{x}{:=}\boldsymbol{B}]$ for $A[x_i{:=}B_i]_{i=1}^{n}$.*

Definition 2.2 (β-reduction) The relation $\to_\beta$ is given by the contraction rule $(\lambda x{:}A_1.A_2)B \to_\beta A_2[x{:=}B]$ and the usual compatibility. $\twoheadrightarrow_\beta$ is the smallest reflexive transitive relation that includes $\to_\beta$; $=_\beta$ is the smallest equivalence relation that includes $\to_\beta$. By $A \twoheadrightarrow_\beta^{+} B$ we indicate that $A \twoheadrightarrow_\beta B$, but $A \not\equiv B$.

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

Definition 2.3 • A *specification* is a triple $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R})$, such that $\boldsymbol{S} \subseteq \mathbb{C}$, $\boldsymbol{A} \subseteq \boldsymbol{S} \times \boldsymbol{S}$ and $\boldsymbol{R} \subseteq \boldsymbol{S} \times \boldsymbol{S} \times \boldsymbol{S}$. The specification is *singly sorted* if $\boldsymbol{A}$ and $\boldsymbol{R}$ are (partial) function from $\boldsymbol{S} \to \boldsymbol{S}$, and $\boldsymbol{S} \times \boldsymbol{S} \to \boldsymbol{S}$ resp. We call $\boldsymbol{S}$ the set of *sorts*, $\boldsymbol{A}$ the set of *axioms*, and $\boldsymbol{R}$ the set of (Π-formation) *rules*.

- A *context* is a finite (maybe empty) list $x_1{:}A_1, \ldots, x_n{:}A_n$ (written $\boldsymbol{x}{:}\boldsymbol{A}$) of variable declarations. $\{x_1, \ldots, x_n\}$ is the *domain* $\textsc{dom}(\boldsymbol{x}{:}\boldsymbol{A})$ of the context. The *empty context* is denoted $\langle\rangle$. We use Γ, Δ as meta-variables for contexts.

Definition 2.4 (Pure Type Systems) Let $\mathfrak{S} = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R})$ be a specification. The Pure Type System $\lambda\mathfrak{S}$ describes how judgements $\Gamma \vdash_{\mathfrak{S}} A : B$ (or $\Gamma \vdash A : B$, if it is clear which $\mathfrak{S}$ is used) can be derived. $\Gamma \vdash A : B$ states that A has type B in context Γ. The typing rules are given in Figure 1.

A context Γ is *legal* if there are A, B such that $\Gamma \vdash A : B$. A term A is

$$\begin{array}{lcl}
\text{(axiom)} & \langle\rangle \vdash s_1 : s_2 & (s_1, s_2) \in \boldsymbol{A} \\[2ex]
\text{(start)} & \dfrac{\Gamma \vdash A : s}{\Gamma, x{:}A \vdash x : A} & x \notin \textsc{dom}\,(\Gamma) \\[2ex]
\text{(weak)} & \dfrac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s}{\Gamma, x{:}C \vdash A : B} & x \notin \textsc{dom}\,(\Gamma) \\[2ex]
(\Pi) & \dfrac{\Gamma \vdash A : s_1 \qquad \Gamma, x{:}A \vdash B : s_2}{\Gamma \vdash (\Pi x{:}A.B) : s_3} & (s_1, s_2, s_3) \in \boldsymbol{R} \\[2ex]
(\lambda) & \dfrac{\Gamma, x{:}A \vdash b : B \qquad \Gamma \vdash (\Pi x{:}A.B) : s}{\Gamma \vdash (\lambda x{:}A.b) : (\Pi x{:}A.B)} & \\[2ex]
\text{(appl)} & \dfrac{\Gamma \vdash F : (\Pi x{:}A.B) \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x{:=}a]} & \\[2ex]
\text{(conv)} & \dfrac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : s \qquad B =_\beta B'}{\Gamma \vdash A : B'} &
\end{array}$$

Fig. 1. The typing rules of PTSs

legal if there are Γ, B such that $\Gamma \vdash A : B$ or $\Gamma \vdash B : A$.

An important class of PTSs is formed by the eight PTSs of the Barendregt Cube [1]. These systems all have $\boldsymbol{S} = \{*, \Box\}$, $\boldsymbol{A} = \{(*{:}\Box)\}$, but differ on $\boldsymbol{R}$.

2.2 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 in Automath as a *line*. Lines are stored in so-called *books*. For writing lines and books in Aut-68 we need: • The symbol `type`; • A set $\mathcal{V}$ of variables; • A set $\mathcal{C}$ of constants; • The symbols ( ) [ ] : — , . We assume $\mathcal{V}$ and $\mathcal{C}$ are infinite, $\mathcal{V} \cap \mathcal{C} = \varnothing$ and $\texttt{type} \notin \mathcal{V} \cup \mathcal{C}$.

Definition 2.5 (Expressions) Define the set $\mathcal{E}$ of Aut-*68-expressions* by:

(variable) If $x \in \mathcal{V}$ then $x \in \mathcal{E}$.

(parameter) If $a \in \mathcal{C}$, $n \in \mathbb{N}$ ($n = 0$ is allowed) 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 $x \in \mathcal{V}$, $\Sigma \in \mathcal{E} \cup \{\texttt{type}\}$ 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 2.6 • The Aut-68-expression $[x{:}\Sigma]\Omega$ is Automath-notation for abstraction terms. In PTS-notation one would write either $\lambda x{:}\Sigma.\Omega$ 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 more complex Automath-systems like aut-QE (see Section 5).

• The Aut-68-expression $\langle\Sigma_2\rangle\Sigma_1$ is Automath-notation for the application of the "function" Σ_1 to the "argument" Σ_2. In PTS-notation: $\Sigma_1\Sigma_2$.[5]

We define $\textsc{fv}(A)$ as for PTSs adding that $\textsc{fv}(a(\Sigma_1,\ldots,\Sigma_n)) \overset{\text{def}}{=} \bigcup_{i=1}^{n} \textsc{fv}(\Sigma_i)$. If $\Omega,\Sigma_1,\ldots,\Sigma_n$ are expressions (in $\mathcal{E}$), and $x_1,\ldots,x_n$ are distinct variables, then $\Omega[x_1,\ldots,x_n{:=}\Sigma_1,\ldots,\Sigma_n]$ denotes the expression Ω (in $\mathcal{E}$) in which all free occurrences of $x_1,\ldots,x_n$ have simultaneously been replaced by $\Sigma_1,\ldots,\Sigma_n$. Correctness of this definition is shown by induction on the structure of Ω. We define $\texttt{type}[x_1,\ldots,x_n{:=}\Sigma_1,\ldots,\Sigma_n]$ as $\texttt{type}$.

Definition 2.7 (Books/lines) An AUT-68-*book* (or *book*) is a finite list (possibly empty) of (AUT-68)-lines. If $l_1,\ldots,l_n$ are the lines of book $\mathfrak{B}$, we write $\mathfrak{B} \equiv l_1,\ldots,l_n$. An AUT-68-*line* (or *line*) is a 4-tuple $(\Gamma; k; \Sigma_1; \Sigma_2)$ where:

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

Remark 2.8 Three sorts of Automath-lines (see Example 2.9):

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

(ii) $(\Gamma; k; \textsc{pn}; \Sigma_2)$ with $k\in\mathcal{C}$. This line introduces a *primitive notion*: A constant k of type Σ_2. Constant k can act as a primitive notion (e.g., introducing the number 0, or the type of natural numbers), or as an axiom. The introduction of k is *parametrised* by the context Γ. For instance, when introducing the primitive notion of "logical conjunction", we do not use a separate primitive notion for each possible conjunction $\texttt{and}(A,B)$. Instead, we use one primitive notion $\texttt{and}$, to which we can add two propositions A and B as parameters when needed to form the proposition $\texttt{and}(A,B)$. Hence, we introduce $\texttt{and}$ in a context $\Gamma\equiv \texttt{x}{:}\texttt{prop},\texttt{y}{:}\texttt{prop}$. Given propositions A,B we can form the AUT-68-expression $\texttt{and}(A,B)$;

(iii) $(\Gamma; k; \Sigma_1; \Sigma_2)$ with $k\in\mathcal{C}$ and $\Sigma_1\in\mathcal{E}$. This line introduces a *definition*. The *definiendum* k is defined by the *definiens* Σ_1 and has *type* Σ_2. Definitions are parametrised like primitive notions. They help to clarify the book structure, make expression manipulations efficient, and abbreviate long expressions by a name. E.g., 7 names $\texttt{S(S(S(S(S(S(S(0)))))))}$.

Example 2.9 In Figure 2 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

[5] Note the unusual *order* of "function" Σ_1 and "argument" Σ_2. The advantages of writing $\langle\Sigma_2\rangle\Sigma_1$ instead of $\Sigma_1\Sigma_2$ are extensively discussed in [21].

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

Fig. 2. Example of an AUTOMATH-book

comments below. To keep things clear, we have omitted the types of the variables in the context. The book consists of three parts:

- In lines 1–5 we introduce some basic material:
  1. The type `prop` (of propositions) is a primitive notion.
  2. We declare a variable `x` of type `prop`. `x` will be used in the book;
  3. We define a variable `y` of type `prop` within the context `x:prop`.
  4. Given propositions `x` and `y`, we introduce a 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 lines 6–11 we show how we can construct proofs of propositions of the form `and`(`x`, `y`), and how we can use proofs of such propositions:
  6. Given propositions `x` and `y`, we assume that we have a `px` $\in \mathcal{V}$ of type `proof(x)`. I.e., the variable `px` represents a proof of `x`;
  7. We also assume a proof `py` of `y`;
  8. Given propositions `x` and `y`, and proofs `px` and `py` of `x` and `y`, we want to conclude that `and(x,y)` holds. This is a natural deduction axiom called `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, `proof(and)` is the type of `and-I` instead of `proof(and(x,y))`. Automath does this to keeps lines short.
  9. To express how we can use a proof of `and(x,y)`, first we introduce a

variable `pxy` that represents an arbitrary proof of `and(x,y)`;

10. As we want `x` to hold whenever `and(x,y)` holds, we introduce an axiom `and-O1` (and-out, first and-elimination). Given propositions `x,y` and a proof `pxy` of the proposition `and(x,y)`, `and-O1(x,y,pxy)` is a proof of `x`;
11. Similarly, we introduce an axiom `and-O2` representing a proof of `y`;

- We can now derive some elementary theorems:

12. We want to derive `and(x,x)` from `x`. I.e., construct a proof of `and(x,x)` from a proof of `x`. In line 6, we introduced a variable `px` for a proof of `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 `prx`, in the context `x`;
13. We derive our theorem: The reflexivity of logical conjunction. Given a proposition `x`, and a proof `prx` of `x`, we can use the axiom `and-I` to find a proof of `and(x,x)`: we can use `and-I(x,x,px,px)` thanks to line 8. We give a name to this proof: `and-R`. If, anywhere in the sequel of the book, Σ is a proposition, and Ω is a proof of Σ, we can write `and-R`(Σ,Ω) for a proof of `and`(Σ,Σ). This is shorter, and more expressive, than `and-I`$(\Sigma,\Sigma,\Omega,\Omega)$;
14. We show `and` is symmetric: Whenever `and(x,y)` holds, we have `and(y,x)`. Given propositions `x,y` and a proof `pxy` of `and(x,y)`, we can form proofs `and-O1(x,y,pxy)` of `x` and `and-O2(x,y,pxy)` of `y`. We feed these proofs "in reverse order" to the axiom `and-I`: `and-I(y,x,and-O2,and-O1)` represents a proof of `and(y,x)`. The expressions `and-O2` and `and-O1` must be read as `and-O2(x,y,pxy)` and `and-O1(x,y,pxy)`.

2.3 Correct books

Not all books are good books. If $(\Gamma;k;\Sigma_1;\Sigma_2)$ is a line of a book $\mathfrak{B}$, the expressions Σ_1 and Σ_2 (as long as Σ_1 is not PN or —, and Σ_2 is not `type`) must be well-defined, i.e. the elements of $\mathcal{V}\cup\mathcal{C}$ occurring in them must have been established (as variables, primitive notions, or defined constants) in earlier parts of $\mathfrak{B}$. The same holds for the type assignments $x_i{:}\alpha_i$ of Γ. Moreover, if Σ_1 is not PN or —, then Σ_1 must be of the same type as k, hence Σ_1 must be of type Σ_2 (within context Γ). Finally, there should be only one definition of any object in a book, so k should not occur in earlier lines. So we need notions of correctness and of typing (with respect to a book and/or a context).

We write $\mathfrak{B};\varnothing\vdash$ OK to indicate that book $\mathfrak{B}$ is correct, and $\mathfrak{B};\Gamma\vdash$ OK to indicate that context Γ is correct with respect to the (correct) book $\mathfrak{B}$.[6] We write $\mathfrak{B};\Gamma\vdash\Sigma_1:\Sigma_2$ to indicate that Σ_1 is a correct expression of type Σ_2 (or simply a correct expression) with respect to $\mathfrak{B}$ and Γ. We also say $\Sigma_1:\Sigma_2$ is a correct *statement* with respect to $\mathfrak{B}$ and Γ. We write $\vdash_{\mathrm{AUT-68}}$ if a confusion of system arises. The following two interrelated definitions are based on [12].

Definition 2.10 (Correct books and contexts) A book $\mathfrak{B}$ and a context Γ are *correct* if $\mathfrak{B};\Gamma\vdash$ OK can be derived with the rules below ($=_{\beta\mathrm{d}}$ is given

[6] As the empty context will be correct with respect to any correct book, this does not lead to misunderstandings.

in Section 2.4. The rules use *correct statements* of Definition 2.11):

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

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

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

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

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

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

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

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

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

Definition 2.11 (Correct statements) A statement $\mathfrak{B};\Gamma\vdash\Sigma:\Omega$ is *correct* if it can be derived with the rules below (the start rule uses the notions of correct context and correct book as given in Definition 2.10).

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

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

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

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

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

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

(conversion)
$$\frac{\mathfrak{B};\Gamma\vdash\Sigma:\Omega_1\qquad\mathfrak{B};\Gamma\vdash\Omega_2{:}\texttt{type}\qquad\mathfrak{B};\Gamma\vdash\Omega_1=_{\beta\text{d}}\Omega_2}{\mathfrak{B};\Gamma\vdash\Sigma:\Omega_2}$$

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

Lemma 2.12 *The book of Example 2.9 (see Figure 2) is correct.*

2.4 Definitional equality

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

Definition 2.13 (d-equality) Let $\mathfrak{B};\Gamma \vdash \Sigma : \Sigma'$. We define the d-*normal form* $\mathrm{nf_d}(\Sigma)$ of Σ with respect to $\mathfrak{B}$ by induction on the length of $\mathfrak{B}$. Assume $\mathrm{nf_d}(\Sigma)$ has been defined for all $\mathfrak{B}'$ with less lines than $\mathfrak{B}$ and all correct Σ with respect to $\mathfrak{B}'$ and a context Γ. By induction on the structure of Σ:

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

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

Definition 2.14 Σ_1 and Σ_2 are called *definitionally equal* (with respect to a book $\mathfrak{B}$) if $\Sigma_1 =_{\beta\mathrm{d}} \Sigma_2$.

Instead of Definition 2.13, d-equality can be given via a reduction relation.

Definition 2.15 (δ-reduction) Let $\mathfrak{B}$ be a book, Γ a correct context with respect to $\mathfrak{B}$, and Σ a correct expression with respect to $\mathfrak{B};\Gamma$. We define $\Sigma \to_\delta \Omega$ by the usual compatibility rules, and

[7] This definition depends on the definition of derivability $\vdash$ which in turn depends on the definition of $=_{\beta\mathrm{d}}$. The definitions of correct book, correct line, correct context, correct expression and $=_{\beta\mathrm{d}}$ should be given within one definition, using induction on the length of the book. This would lead to a correct but very long definition, and that is the reason why the definitions are split into smaller parts (in this paper as well as in [12]).

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

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

We say that Σ is in δ-normal form if for no expression Ω, $\Sigma \to_\delta \Omega$, and define $\twoheadrightarrow_\delta$, $\twoheadrightarrow_\delta^+$ and $=_\delta$ as usual. $\to_\delta$ depends on $\mathfrak{B}$, but as before, we drop $\mathfrak{B}$ if no confusion occurs. The relations $=_{\mathrm{d}}$ and $=_\delta$ are the same:

Lemma 2.16 *1• (Church-Rosser) If $A_1 =_\delta A_2$ then there is B such that $A_1 \twoheadrightarrow_\delta B$ and $A_2 \twoheadrightarrow_\delta B$.* *2• $\mathrm{nf}_{\mathrm{d}}(\Sigma)$ is the unique δ-normal form of Σ.* *3• $\Sigma =_\delta \Omega$ if and only if $\Sigma =_{\mathrm{d}} \Omega$.* *4• $\to_\delta$ is strongly normalising.*

Definition 2.17 • A book $\mathfrak{B}$ is part of a book $\mathfrak{B}'$, denoted as $\mathfrak{B} \subseteq \mathfrak{B}'$, if all lines of $\mathfrak{B}$ are lines of $\mathfrak{B}'$.

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

Lemma 2.18 (Weakening) *If $\mathfrak{B};\Gamma \vdash \Sigma : \Omega$, $\mathfrak{B} \subseteq \mathfrak{B}'$, $\Gamma \subseteq \Gamma'$ and $\mathfrak{B}';\Gamma' \vdash$ ok then $\mathfrak{B}';\Gamma' \vdash \Sigma : \Omega$.*

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

To describe Aut-68 as a PTS $\lambda 68$, we translate Aut-68-expressions to λ-terms:

Definition 3.1 Recall that $\mathbb{T}$ and $\mathbb{V}$ are the set of terms and variables for PTSs. We define a mapping $\overline{[\ldots]}$ from the correct expressions in $\mathcal{E}$ (relative to a book $\mathfrak{B}$ and a context Γ) to $\mathbb{T}$. We assume that $\mathcal{C} \cup \mathcal{V} \subseteq \mathbb{V}$.
• $\overline{x} \overset{\text{def}}{=} x$ for $x \in \mathcal{V}$; • $\overline{b(\Sigma_1,\ldots,\Sigma_n)} \overset{\text{def}}{=} b\overline{\Sigma_1}\cdots\overline{\Sigma_n}$; • $\overline{\langle\Omega\rangle\Sigma} \overset{\text{def}}{=} \overline{\Sigma}\,\overline{\Omega}$; • $\overline{\mathtt{type}} \overset{\text{def}}{=} *$;
• $\overline{[x{:}\Sigma]\Omega} \overset{\text{def}}{=} \Pi x{:}\overline{\Sigma}.\overline{\Omega}$ if $[x{:}\Sigma]\Omega$ has type $\mathtt{type}$, otherwise $\overline{[x{:}\Sigma]\Omega} \overset{\text{def}}{=} \lambda x{:}\overline{\Sigma}.\overline{\Omega}$;

With this translation in mind, we want to find a type system $\lambda 68$ that "suits" Aut68, i.e. if Σ is a correct expression of type Ω with respect to a book $\mathfrak{B}$ and a context Γ, then we want $\mathfrak{B}',\Gamma' \vdash \overline{\Sigma} : \overline{\Omega}$ to be derivable in $\lambda 68$, and vice versa. Here, $\mathfrak{B}'$ and Γ' are some suitable translations of $\mathfrak{B}$ and Γ. The search for a suitable $\lambda 68$ will focus on three points: Π-formation and parameter types; constants and variables; and definitions.

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

As $\overline{\mathtt{type}} \equiv *$, Definition 2.11 clarifies which Π-rules are implied by the abstraction mechanism of Aut-68:

The rule $\dfrac{\mathfrak{B};\Gamma \vdash \Sigma_1{:}\mathtt{type} \qquad \mathfrak{B};\Gamma, x{:}\Sigma_1 \vdash \Omega_1{:}\mathtt{type}}{\mathfrak{B};\Gamma \vdash [x{:}\Sigma_1]\Omega_1 : \mathtt{type}}$

translates into the PTSs Π-rule $(*,*,*)$ $\dfrac{\overline{\mathfrak{B}},\overline{\Gamma} \vdash \overline{\Sigma_1}{:}\, * \qquad \overline{\mathfrak{B}},\overline{\Gamma}, x{:}\overline{\Sigma_1} \vdash \overline{\Omega_1}{:}*}{\overline{\mathfrak{B}},\overline{\Gamma} \vdash (\Pi x{:}\overline{\Sigma_1}.\overline{\Omega_1}) : *}$

It is, however, not immediately clear which Π-rules are induced by the parameter mechanism of Aut-68. Let $\Sigma \equiv b(\Sigma_1,\ldots,\Sigma_n)$ be a correct expression of type Ω with respect to a book $\mathfrak{B}$ and a context Γ. By Definition 2.10 there is a line $(x_1{:}\alpha_1,\ldots,x_n{:}\alpha_n;b;\Xi_1;\Xi_2)$ in $\mathfrak{B}$ such that each Σ_i is a correct expression with respect to $\mathfrak{B}$ and Γ, and has a type that is definitionally equal to $\alpha_i[x_1,\ldots,x_{i-1}{:=}\Sigma_1,\ldots,\Sigma_{i-1}]$. We also know that $\Omega =_{\beta\mathrm{d}} \Xi_2[x_1,\ldots,x_n{:=}\Sigma_1,\ldots\Sigma_n]$. Now $\overline{\Sigma} \equiv b\overline{\Sigma_1}\cdots\overline{\Sigma_n}$, and, assuming that we can derive in $\lambda 68$ that $\overline{\Sigma_i}$ has type $\overline{\alpha_i}[x_1,\ldots,x_{i-1}{:=}\overline{\Sigma_1},\ldots,\overline{\Sigma_{i-1}}]$, it is not unreasonable to assign the type $\Pi x_1{:}\overline{\alpha_1}\cdots\Pi x_n{:}\overline{\alpha_n}to\mathrm{b}.\overline{\Xi_2}$. We will abbreviate this last term by $\prod_{i=1}^n x_i{:}\overline{\alpha_i}.\overline{\Xi_2}$. Then we can derive (using n times the application rule that we will introduce for $\lambda 68$) that $\overline{\Sigma}$ has type $\overline{\Omega}$ in $\lambda 68$.

It is important to notice that the type of b, $\prod_{i=1}^n x_i{:}\overline{\alpha_i}.\overline{\Xi_2}$, does not necessarily have an equivalent in AUT-68, as in AUT-68 abstractions over `type` are not allowed (only abstractions over expressions Σ that have `type` as type are possible — cf. Definition 2.11). In other words, the type of b, $\prod_{i=1}^n x_i{:}\overline{\alpha_i}.\overline{\Xi_2}$, is not necessarily a first-class citizen of AUT-68 and should therefore have special treatment in $\lambda 68$. This is the reason to create a special sort $\triangle$, in which these types of AUT-68 constants and definitions are stored. This idea originates from van Benthem Jutting and was firstly presented in [1].

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

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

In Example 5.2.4.8 of [1], there is no rule $(*,*,\triangle)$. In principle, this rule is superfluous, as each application of rule $(*,*,\triangle)$ can be replaced by an application of rule $(*,*,*)$. Nevertheless we maintain this rule as:

- The presence of both $(*,*,*)$ and $(*,*,\triangle)$ in the system stresses the fact that AUT-68 has two type mechanisms: One provided by the parameter mechanism and one by the λ-abstraction mechanism;
- There are technical arguments to make a distinction between types formed by the abstraction mechanism and types that appear via the parameter mechanism. In this paper, we denote product types constructed by the abstraction mechanism in the usual way (so: $\Pi x{:}A.B$), whilst we will use the notation $\P x{:}A.B$ for a type constructed by the parameter mechanism. Hence, we have for the constant b above that $b : \P_{i=1}^n x_i{:}\overline{\alpha_i}.\overline{\Xi_2}$[9]. As an additional advantage, the resulting system will maintain Unicity of Types.

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

This would have been lost if we use rules $(*,*,*)$ and $(*,*,\triangle)$ without making this difference, as we can then by these rules derive both:

$$\frac{\alpha{:}* \vdash \alpha{:}\ * \qquad \alpha{:}*, x{:}\alpha \vdash \alpha{:}*}{\alpha{:}* \vdash (\Pi x{:}\alpha.\alpha) : *} \text{ and } \frac{\alpha{:}* \vdash \alpha{:}\ * \qquad \alpha{:}*, x{:}\alpha \vdash \alpha{:}*}{\alpha{:}* \vdash (\Pi x{:}\alpha.\alpha) : \triangle}$$

3.2 The different treatment of constants and variables

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

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

This process can be captured by rule: $\dfrac{\Delta;\overline{\Gamma} \vdash \overline{\Xi_2}{:}s_1 \qquad \Delta; \vdash \P\,\overline{\Gamma}.\overline{\Xi_2}{:}s_2}{\Delta, b{:}\,\P\,\overline{\Gamma}.\overline{\Xi_2}; \vdash b{:}\,\P\,\overline{\Gamma}.\overline{\Xi_2}}$.
Here $s_1 \in \{*, \square\}$ (compare: $\Xi_2{:}\texttt{type}$ or $\Xi_2 \equiv \texttt{type}$) and $s_2 \in \{*, \square, \triangle\}$ (usually, $s_2 \equiv \triangle$; the cases $s_2 \equiv *, \square$ only occur if Γ is empty).

3.3 The definition system and the translation using §

A line $(x_1{:}\alpha_1, \ldots, x_n{:}\alpha_n; b; \Xi_1; \Xi_2)$, in which b is a constant and $\Xi_1 \in \mathcal{E}$, represents the definition: "For all expressions $\Omega_1, \ldots, \Omega_n$ (obeying some type conditions), $b(\Omega_1, \ldots, \Omega_n)$ abbreviates $\Xi_1[x_1, \ldots, x_n{:=}\Omega_1, \ldots, \Omega_n]$, and has type $\Xi_2[x_1, \ldots, x_n{:=}\Omega_1, \ldots, \Omega_n]$." So in $\lambda 68$, the context should have $bX_1 \cdots X_n$ "is equal to" $\Xi_1[x_1, \ldots, x_n{:=}X_1, \ldots, X_n]$, for all terms $X_1, \ldots, X_n$. This can be done by writing $b{:=}\left(\lambda_{i=1}^{n}\, x_i{:}\overline{\alpha_i}.\overline{\Xi_1}\right) : \left(\P_{i=1}^{n}\, x_i{:}\overline{\alpha_i}.\overline{\Xi_2}\right)$ in the context instead of only $b{:}\,\P_{i=1}^{n}\, x_i{:}\overline{\alpha_i}.\overline{\Xi_2}$, and adding a δ-reduction rule which unfolds the definition of b: $\Delta \vdash b \to_\delta \lambda_{i=1}^n x_i{:}\overline{\alpha_i}.\overline{\Xi_1}$ whenever $b{:=}\left(\lambda_{i=1}^n x_i{:}\overline{\alpha_i}.\overline{\Xi_1}\right) : \left(\P_{i=1}^n x_i{:}\overline{\alpha_i}.\overline{\Xi_2}\right) \in \Delta$. Unfolding the definition of b in a term $b\overline{\Sigma_1}\cdots\overline{\Sigma_n}$ and applying β-reduction n times gives $\overline{\Xi_1}[x_1{:=}\overline{\Sigma_1}]\cdots[x_n{:=}\overline{\Sigma_n}]$. In AUT-68[10], this corresponds to $\Delta \vdash b(\Sigma_1,\ldots,\Sigma_n) \to_\delta \Xi_1[x_1,\ldots,x_n{:=}\Sigma_1,\ldots,\Sigma_n]$.

This method, however, has disadvantages:

- In the AUT-68 line $(x_1{:}\alpha_1,\ldots,x_n{:}\alpha_n; b; \Xi_1; \Xi_2)$, $b(\Sigma_1,\ldots,\Sigma_n)$ has $b\overline{\Sigma_1}\cdots\overline{\Sigma_n}$ as its equivalent in $\lambda68$. If $n>0$, the latter $\lambda68$-term has $B \equiv b\overline{\Sigma_1}\cdots\overline{\Sigma_m}$ as a subterm for any $m<n$. But B has no equivalent in AUT-68: Only after B is applied to suitable terms $\overline{\Sigma_{m+1}},\ldots,\overline{\Sigma_n}$ the result $B\overline{\Sigma_{m+1}}\cdots\overline{\Sigma_n}$ has $b(\Sigma_1,\ldots,\Sigma_n)$ as its equivalent in AUT-68. Hence B must not be seen as a term directly translatable into AUTOMATH, but only as an intermediate result necessary to construct the equivalent of $b(\Sigma_1,\ldots,\Sigma_n)$. B is recognisable as an intermediate result via its type $\P_{i=m+1}^n x_i{:}\overline{\alpha_i}.\overline{\Xi_2}$, of sort $\triangle$ (not $*$ or $\Box$).

  The method above allows to unfold the definition of b in B, because $b\overline{\Sigma_1}\cdots\overline{\Sigma_m}$ can reduce to $\left(\lambda_{i=1}^n x_i{:}\overline{\alpha_i}.\overline{\Xi_1}\right)\overline{\Sigma_1}\cdots\overline{\Sigma_m}$, and we can β-reduce this term m times to $\left(\lambda_{i=m+1}^n x_i{:}\overline{\alpha_i}.\overline{\Xi_1}\right)[x_j{:=}\overline{\Sigma_j}]_{j=1}^m$. In AUT-68 such unfolding is not possible before *all* n arguments $\overline{\Sigma_1},\ldots,\overline{\Sigma_n}$ are applied to b, so only when the construction of the equivalent of $b(\Sigma_1,\ldots,\Sigma_n)$ has been completed;
- $\lambda_{i=1}^n x_i{:}\overline{\alpha_i}.\overline{\Xi_1}$ does not necessarily have an equivalent in AUT-68. Consider for instance the constant b in the line $(\alpha{:}\texttt{type}; b; [x{:}\alpha]x; [x{:}\alpha]\alpha)$. In this case, $\lambda_{i=1}^n x_i{:}\overline{\alpha_i}.\overline{\Xi_1} \equiv \lambda\alpha{:}*.\lambda x{:}\alpha.x$. Its equivalent in AUT-68 is $[\alpha{:}\texttt{type}][x{:}\alpha]x$, but an abstraction $[\alpha{:}\texttt{type}]$ cannot be made in AUT-68.[11] This is the reason why we do not incorporate $\lambda_{i=1}^n x_i{:}\overline{\alpha_i}.\overline{\Xi_1}$ as a citizen of $\lambda68$.

Hence we choose another translation. The line $(x_1{:}\alpha_1,\ldots,x_n{:}\alpha_n; b; \Xi_1; \Xi_2)$, where $\Xi_1 \in \mathcal{E}$, is translated by taking $b{:=}\left(\S_{i=1}^n x_i{:}\overline{\alpha_i}.\overline{\Xi_1}\right) : \left(\P_{i=1}^n x_i{:}\overline{\alpha_i}.\overline{\Xi_2}\right)$ instead of $b{:=}\left(\lambda_{i=1}^n x_i{:}\overline{\alpha_i}.\overline{\Xi_1}\right) : \left(\P_{i=1}^n x_i{:}\overline{\alpha_i}.\overline{\Xi_2}\right)$ in the left part of the context. A reduction rule $bX_1\cdots X_n \to_\delta \overline{\Xi_1}[x_1,\ldots,x_n{:=}X_1,\ldots,X_n]$ is added for all terms $X_1,\ldots,X_n$. We use $\S$ instead of λ to emphasise that, though both $\S x{:}A$ and $\lambda x{:}A$ are abstractions, they are not the same kind of abstraction.

4 $\lambda68$

Here, we give $\lambda68$, show that it has the desirable properties of PTSs and that it is the PTS version of AUT-68.

Definition 4.1 ($\lambda68$)

(i) Terms of $\lambda68$ are given by $\mathcal{T} ::= \mathcal{V} \mid \mathcal{C} \mid \boldsymbol{S} \mid \mathcal{T}\mathcal{T} \mid \lambda\mathcal{V}{:}\mathcal{T}.\mathcal{T} \mid \S\mathcal{V}{:}\mathcal{T}.\mathcal{T} \mid \Pi\mathcal{V}{:}\mathcal{T}.\mathcal{T} \mid \P\mathcal{V}{:}\mathcal{T}.\mathcal{T}$, where $\boldsymbol{S}$ is the set of sorts $\{*, \Box, \triangle\}$. Free variables

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

[11] Compare with the situation of Section 3.1, where we found that the type of b is not necessarily a first-class citizen of AUT-68. There, we could not avoid that the type of b became a citizen of $\lambda68$ (though we made it second-class by storing it in the sort $\triangle$).

$\textsc{fv}(T)$ and "free" constants $\textsc{fc}(T)$ of term T are defined as usual;

(ii) We define the notion of context inductively:

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

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

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

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

(iv) We use the usual notion of β-reduction;

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

Notice the lack of rule (§) as we do not want that terms of the form $\S\, x{:}A.B$ be first-class citizens of $\lambda 68$: they do not have an equivalent in Automath.

Example 4.2 The translation of Example 2.9 into $\lambda 68$ is given in Figure 4.[12] We see 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 part of the context, instead of the left part.

[12] Because of the habit in computer science to use more than one digit for a variable, we have to write additional brackets around subterms like `proof` to keep things unambiguous.

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

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

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

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

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

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

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

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

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

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

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

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

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

Fig. 3. Rules of $\lambda 68$

Lemma 4.3 (Free Variable Lemma)
For $\Delta;\Gamma \vdash M : N$, $\Delta \equiv b_1{:}B_1, \cdots, b_m{:}B_m$ and $\Gamma \equiv x_1{:}A_1, \ldots, x_n{:}A_n$ [13] *:*

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

Lemma 4.4 • **(Start)** *Let $\Delta;\Gamma$ be a legal context. Then $\Delta;\Gamma \vdash * : \Box$, and if $b{:}A \in \Delta;\Gamma$, or $c{:=}T{:}A \in \Delta$, then $\Delta;\Gamma \vdash c : A$.*

- **(Definition)** *Let* $\Delta_1, b{:=}\left(\S_{i=1}^{n} x_i{:}A_i.T\right){:}\left(\P_{i=1}^{n} x_i{:}A_i.B\right), \Delta_2;\Gamma \vdash M : N$, *where $B \not\equiv \P y{:}B_1.B_2$. Then $\Delta_1; x_1{:}A_1, \ldots, x_n{:}A_n \vdash T : B : s$ for $s \in \{*, \Box\}$.*

[13] In Δ, also expressions $b_i{:=}T_i{:}B_i$ may occur, but for uniformity we leave out the $:=T_i$-part.

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

Fig. 4. Translation of Example 2.9

Definition 4.5 We define: $\Delta_1;\Gamma_1 \vdash \Delta_2;\Gamma_2$ if and only if

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

Lemma 4.6 • **(Transitivity)** *Assume* $\Delta_1;\Gamma_1 \vdash \Delta_2;\Gamma_2$ *and* $\Delta_2;\Gamma_2 \vdash B : C$. *Then* $\Delta_1;\Gamma_1 \vdash B : C$.

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

Lemma 4.7 (Generation Lemma)

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

Lemma 4.8 • **(Unicity of Types)** *If* $\Delta;\Gamma \vdash A : B_1$ *and* $\Delta;\Gamma \vdash A : B_2$ *then* $B_1 =_{\beta\delta} B_2$.

- **(Correctness of Types)** *If* $\Delta;\Gamma \vdash A : B$ *then there is* $s \in \boldsymbol{S}$ *such that* $B \equiv s$ *or* $\Delta;\Gamma \vdash B : s$.

- *If* $\Delta;\Gamma \vdash A : (\Pi x{:}B_1.B_2)$ *then* $\Delta;\Gamma \vdash B_1 : *$*; and* $\Delta;\Gamma, x{:}B_1 \vdash B_2 : *$.
- *If* $\Delta;\Gamma \vdash A : (\P x{:}B_1.B_2)$ *then* $\Delta;\Gamma \vdash B_1 : s_1$ *for* $s_1 \in \{*, \Box\}$*;*
  and $\Delta;\Gamma, x{:}B_1 \vdash B_2{:}s_2$ *for some* s_2.

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

Definition 4.9 • Let Γ be a Aut-68-context $x_1{:}\alpha_1, \ldots, x_n{:}\alpha_n$. Then $\overline{\Gamma} \overset{\text{def}}{=} x_1{:}\overline{\alpha_1}, \ldots, x_n{:}\overline{\alpha_n}$.

- Let $\mathfrak{B}$ be a book. We define the left part $\overline{\mathfrak{B}}$ of a context in λ68:
  - $\overline{\varnothing} \overset{\text{def}}{=} \varnothing$;
  - $\overline{\mathfrak{B}, (\Gamma; b; \text{PN}; \Omega)} \overset{\text{def}}{=} \overline{\mathfrak{B}}, b{:}\ \P\,\overline{\Gamma}.\overline{\Omega}$;
  - $\overline{\mathfrak{B}, (\Gamma; x; \text{—}; \Omega)} \overset{\text{def}}{=} \overline{\mathfrak{B}}$;
  - $\overline{\mathfrak{B}, (\Gamma; b; \Sigma; \Omega)} \overset{\text{def}}{=} \overline{\mathfrak{B}}, b{:=}\ \S\,\overline{\Gamma}.\overline{\Sigma}{:}\ \P\,\overline{\Gamma}.\overline{\Omega}$.

Lemma 4.10 *Assume,* Σ *is a correct expression with respect to a book* $\mathfrak{B}$.
- *1.* $\Sigma \rightarrow_\beta \Sigma'$ *if and only if* $\overline{\Sigma} \rightarrow_\beta \overline{\Sigma'}$*;*
- *2.* $\mathfrak{B} \vdash_{AUT-68} \Sigma \rightarrow_\delta \Sigma'$ *if and only if* $\overline{\mathfrak{B}} \vdash_{\lambda 68} \overline{\Sigma} \rightarrow_\delta \overline{\Sigma'}$.

Theorem 4.11 (Church-Rosser for $\rightarrow_{\beta\delta}$) *Let* Δ *be the left part of a context in which* M *is typable. If* $\Delta \vdash M \twoheadrightarrow_{\beta\delta} N_1$ *and* $\Delta \vdash M \twoheadrightarrow_{\beta\delta} N_2$ *then there is* P *such that* $\Delta \vdash N_1 \twoheadrightarrow_{\beta\delta} P$ *and* $\Delta \vdash N_2 \twoheadrightarrow_{\beta\delta} P$.

Lemma 4.12 (Subject Reduction) *Let* $\Delta;\Gamma \vdash A : B$.
1. If $A \rightarrow_\beta A'$ *then* $\Delta;\Gamma \vdash A' : B$. *2.* $A \rightarrow_\delta A'$ *then* $\Delta;\Gamma \vdash A' : B$. *3. If* $A \twoheadrightarrow_{\beta\delta} A'$ *then* $\Delta;\Gamma \vdash A' : B$.

Lemma 4.13 *Assume* $s \in \boldsymbol{S}$ *and* M *legal. Then* $(\Delta \vdash M =_{\beta\delta} s) \Rightarrow M \equiv s$.

Theorem 4.14 (Strong Normalisation) λ68 *is* $\beta\delta$*-strongly normalising.*

The next two theorems formally relate Aut-68 and λ68.

Theorem 4.15 *Let* $\mathfrak{B}$ *be an* Automath *book and* Γ *an* Automath *context.*
- *If* $\mathfrak{B};\Gamma \vdash_{AUT-68}$ OK *then* $\overline{\mathfrak{B}};\overline{\Gamma}$ *is legal;*
- *If* $\mathfrak{B};\Gamma \vdash_{AUT-68} \Sigma : \Omega$ *then* $\overline{\mathfrak{B}};\overline{\Gamma} \vdash_{\lambda 68} \overline{\Sigma} : \overline{\Omega}$.

Theorem 4.16 *Let* $\Delta;\Gamma \vdash_{\lambda 68} M : N$. *There is an* Automath *book* $\mathfrak{B}$ *and an* Automath *context* Γ' *such that* $\mathfrak{B};\Gamma' \vdash_{AUT-68}$ OK*, and* $\overline{\mathfrak{B}}, \overline{\Gamma'} \equiv \Delta;\Gamma$. *Also,*

(i) *If* $N \equiv \Box$ *then* $M \equiv *$*;*

(ii) *If* $\Delta;\Gamma \vdash_{\lambda 68} N : \Box$ *then* $N \equiv *$ *and there is* $\Omega \in \mathcal{E}$ *such that* $\overline{\Omega} \equiv M$ *and* $\mathfrak{B};\Gamma' \vdash_{AUT-68} \Omega : \texttt{type}$*;*

(iii) *If* $N \equiv \triangle$ *then there is* $\Gamma'' \equiv x_1{:}\Sigma_1, \ldots, x_n{:}\Sigma_n$, $\Omega \in \mathcal{E} \cup \{\texttt{type}\}$ *with:* • Γ', Γ'' *is correct with respect to* $\mathfrak{B}$*;* • $M \equiv \P\,\overline{\Gamma''}.\overline{\Omega}$*;* • $\Omega \equiv \texttt{type}$ *or* $\mathfrak{B};\Gamma' \vdash_{AUT-68} \Omega : \texttt{type}$*;*

(iv) *If* $\Delta;\Gamma \vdash_{\lambda 68} N : \triangle$ *then there are* $b \in \mathcal{C}$ *and* $\Sigma_1, \ldots, \Sigma_n \in \mathcal{E}$ *such that* $M \equiv b\overline{\Sigma_1} \cdots \overline{\Sigma_n}$. *Moreover,* $\mathfrak{B}$ *contains a line* $(x_1{:}\Omega_1, \ldots, x_m{:}\Omega_m; b; \Xi_1; \Xi_2)$ *such that:* • $N \equiv \left(\P_{i=n+1}^{m} x_i{:}\overline{\Omega_i}.\overline{\Xi_2}\right)[x_1, \ldots, x_n{:=}\overline{\Sigma_1}, \ldots, \overline{\Sigma_n}]$*;* • $m > n$*;* • $\mathfrak{B};\Gamma' \vdash_{AUT-68} \Sigma_i{:}\Omega_i[x_1, \ldots, x_{i-1}{:=}\Sigma_1, \ldots, \Sigma_{i-1}]$ $(1 \leq i \leq n)$*;*

(v) *If* $N \equiv *$ *then there is* $\Omega \in \mathcal{E}$ *where* $\overline{\Omega} \equiv M$ *and* $\mathfrak{B}; \Gamma' \vdash_{AUT-68} \Omega : \texttt{type}$*;*

(vi) *If* $\Delta; \Gamma \vdash_{\lambda 68} N : *$ *then there are* $\Sigma, \Omega \in \mathcal{E}$ *such that* $\overline{\Sigma} \equiv M$ *and* $\overline{\Omega} \equiv N$*, and* $\mathfrak{B}; \Gamma' \vdash_{AUT-68} \Sigma : \Omega$*, and* $\mathfrak{B}; \Gamma' \vdash_{AUT-68} \Omega : \texttt{type}$*.*

5 Conclusion

The system AUT-68 is one of several AUTOMATH-systems. Another frequently used system is AUT-QE. We shall briefly compare AUT-68 to AUT-QE and describe how we can easily adapt $\lambda 68$ to a system λQE. The system AUT-QE has many similarities with AUT-68 but differs on the following extensions:

(i) In AUT-QE we can also form the abstraction expression $[x{:}\Sigma]\texttt{type}$ (thus extending Definition 2.5);

(ii) Inhabitants of types $[x{:}\Sigma]\texttt{type}$ are introduced in AUT-QE by extending abstraction rules 1 and 2 of Definition 2.11 with the AUT-QE rule: $\dfrac{\mathfrak{B}; \Gamma \vdash \Sigma_1{:}\texttt{type} \qquad \mathfrak{B}; \Gamma, x{:}\Sigma_1 \vdash \Sigma_2{:}\texttt{type}}{\mathfrak{B}; \Gamma \vdash [x{:}\Sigma_1]\Sigma_2 : [x{:}\Sigma_1]\texttt{type}}$. Like $\texttt{type}$, $[x{:}\Sigma_1]\texttt{type}$ is not typable. In a translation to a PTS, these expressions should get type $\Box$;

(iii) In AUT-QE, there is a new reduction $\rightarrow_{\text{QE}}$ on expressions, given by the rule $[x_1{:}\Sigma_1]\cdots[x_n{:}\Sigma_n][y{:}\Omega]\texttt{type} \rightarrow_{\text{QE}} [x_1{:}\Sigma_1]\cdots[x_n{:}\Sigma_n]\texttt{type}$ (for $n \geq 0$).

The first two rules are straightforward. They correspond to an extension of $\lambda\!\rightarrow$ to λP in PTSs. It is easy to extend $\lambda 68$ with similar rules; just add the Π-formation rule $(*, \Box, \Box)$: $\dfrac{\Delta; \Gamma \vdash A : * \qquad \Delta; \Gamma, x{:}A \vdash B : \Box}{\Delta; \Gamma \vdash (\Pi x{:}A.B) : \Box}$. The third rule is unusual. It is needed because AUT-QE does not distinguish λs and Πs. In AUT-68 this did not matter, as we could always derive whether $[x{:}\Sigma]\Omega$ should be interpreted as $\lambda x{:}\Sigma.\Omega$ or as $\Pi x{:}\Sigma.\Omega$. The latter should have type $\texttt{type}$, and the first should not have type $\texttt{type}$. Though $\lambda 68$ does not have Π-conversion, it is easy to extend it to a system $\lambda\Pi 68$ following the lines of [18] by:

- Changing rule (App$_1$) into $\dfrac{\Delta; \Gamma \vdash M : \Pi x{:}A.B \qquad \Delta; \Gamma \vdash N : A}{\Delta; \Gamma \vdash MN : (\Pi x{:}A.B)N}$;
- Adding a new reduction rule $\rightarrow_\Pi$ by $(\Pi x{:}A.B)N \rightarrow_\Pi B[x{:=}N]$.

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 [1,15], we feel our description is more accurate and unlike [1,15], pays attention to the definition and parameter systems, which are crucial in AUTOMATH. We provided a PTS called $\lambda 68$ which we showed to be the system AUT-68 written as a PTS. Although $\lambda 68$ does not include Π-conversion (while AUTOMATH does), it is easy to adapt $\lambda 68$ to include Π-conversion following the lines of [18].

The adaptation of $\lambda 68$ to a system λQE, representing the AUTOMATH-system AUT-QE is not hard, either: It requires adapting of the Π-formation rule to include not only the rule $(*, *, *)$ but also $(*, \Box, \Box)$ and the introduction of the additional reduction rule of type inclusion. We leave this as a future work.

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

When comparing $\lambda 68$ to other type systems with definitions, we find an important difference. In $\lambda 68$, the correspondence between types of definiendum and definiens differs from that of the systems in [26,18]. Automath allows *parameters* to occur in the definiens, and there is no parameter mechanism in the PTSs of [1,26,18] althrough this mechansim exists in [22,19,20].

References

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

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

[3] L.S. van Benthem Jutting. Description of AUT-68. Technical Report 12, Eindhoven University of Technology, 1981. Also in [24], pp. 251–273.

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

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

[6] N.G. de Bruijn. The mathematical language AUTOMATH, its usage and some of its extensions. In M. Laudet, D. Lacombe, and M. Schuetzenberger, editors, *Symposium on Automatic Demonstration*, pages 29–61, IRIA, Versailles, 1968. Springer Verlag, Berlin, 1970. Lecture Notes in Mathematics **125**; also in [24].

[7] N.G. de Bruijn. The Mathematical Vernacular, a language for mathematics with typed sets. In P. Dybjer et al., editors, *Proceedings of the Workshop on Programming Languages.* Marstrand, Sweden, 1987. Reprinted in [24].

[8] N.G. de Bruijn. Reflections on Automath. Eindhoven University of Technology, 1990. Also in [24], pages 201–228.

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

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

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

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

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

[14] G. Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens.* Nebert, Halle, 1879. Also in [16], pages 1–82.

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

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

[17] W.A. Howard. The formulas-as-types notion of construction. In [25], pages 479–490, 1980.

[18] F. Kamareddine, R. Bloo, and R. Nederpelt. On π-conversion in the λ-cube and the combination with abbreviations. *Annals of Pure and Applied Logics*, 97:27–45, 1999.

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

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

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

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

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

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

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

[26] P. Severi and E. Poll. Pure type systems with definitions. In A. Nerode and Yu.V. Matiyasevich, editors, *Proceedings of LFCS'94 (LNCS* **813***)*, pages 316–328, New York, 1994. LFCS'94, St. Petersburg, Russia, Springer.

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

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

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