Semantics for Classical AUTOMATH and Related Systems*

HENK Barendregt

Mathematical Institute, University of Utrecht,
Budapestlaan 6; 3508 TA Utrecht, The Netherlands

AND

ADRIAN REZUS

Furkabaan 680, 3524 ZL Utrecht, The Netherlands

INTRODUCTION

Developed from ideas of N. G. de Bruijn (1967) at the Eindhoven University of Technology (The Netherlands), the Automated Mathematics Project is a programme of formalization of actual mathematical texts in view of computer-assisted proof-checking (cf. [5, 6, 11, 21]).

Leaving aside the underlying pragmatic motivations [5, 21], the main languages in the AUT(OMATH)-family may be, roughly, viewed, as being applied typed lambda-calculi with a generalized type structure: a legal AUT-type may often depend on parameters on which its "inhabitants" also depend, such that the AUT-types cannot be characterized "beforehand" (as is the case in the First- and Second-Order Typed Lambda-Calculi [9, 8, 20]).

In [21] one of the authors complained of the lack of formal semantics for the main AUT-languages, briefly surveying the epistemological status of the problem. The present paper is intended to fill in this gap, providing a "mathematical" model-theory for Classical AUTOMATH (CA for short, otherwise called "AUT-68"; see [21] for a detailed description). The main work relies on suggestions given in [24] and consists, essentially, of a "translation" of the type-distinctions of CA into a type-free setting, viz., into specific models of the type-free lambda-calculus [2, 3].

The proposed semantics covers, obviously, the First-order Typed Lambda-Calculus of Church [2] and the analogue Theory of Functionality of Curry [9] and extends, almost trivially, to more involved type-structures as, e.g.,

* The work of the second author was partly completed during his stay at the Department of Mathematics and Computing Science, Eindhoven University of Technology, The Netherlands.
those present in the Second-Order Typed Lambda-Calculus of Girard [8] and Reynolds [20] or in Pure LCF ([18], etc.). However, somewhat "more structure" is necessary in order to interpret—essentially along the same lines—Zucker's AUT-Pi system of [27] or Martin-Löf's Intuitionistic Theory of Types (with one universe; see [16, 17] and [1,4] for alternative semantics of the latter), topics which will be discussed in detail elsewhere.

1. Closure Operations in Additive Domains

Let $D$, $D'$, $D''$, ..., range over complete lattices. For $D$ fixed arbitrarily, $\subseteq_D$ stands for the underlying partial order and $\text{sup}_D X$ denotes the supremum of $X \subseteq D$. A set $X \subseteq D$ is directed if every finite $Y \subseteq X$ has an upper bound in $X$. A map $f: D \to D'$ is continuous if it preserves suprema of directed sets, i.e., $f(\text{sup}_D X) = \text{sup}_D \{f(x): x \in X\}$, for directed $X \subseteq D$. (This is, in fact, topological continuity relative to the so-called Scott-topology; see [22; 7, Chap. II].) The cartesian product $D \times D'$ of two complete lattices consists of ordered pairs partially ordered "componentwise": $(d_1, d_2) \sqsubseteq (d_1', d_2')$ iff $d_1 \subseteq_D d_1'$ and $d_2 \subseteq_D d_2'$. The function space $[D \to D']$ consists of all continuous $f: D \to D'$, with the pointwise ordering: $f \sqsubseteq g$ iff $\forall x \in D$. $f(x) \sqsubseteq_D g(x)$. Obviously, these constructions provide new complete lattices from old.

1.1. Proposition. (i) A map $f: D \times D' \to D''$ is continuous if and only if it is continuous in each variable separately.

(ii) The map $\text{ev}: [D \to D'] \times D \to D'$, defined by $\text{ev}(f, x) = f(x)$, is continuous.

(iii) Let $f \in [D \times D' \to D'']$. Then the map $\tilde{f}: D \to [D' \to D'']$, defined by $\tilde{f}(x) = \lambda y \cdot f(x, y)$ is continuous. Moreover, the map $\text{abs} = \lambda f \cdot \tilde{f}$ is continuous.

Proof. Well known; cf. [22] or [7].

It follows that the category of complete lattices with continuous maps as morphisms is cartesian closed.

1.2. Proposition. Every $f \in [D \to D]$ has a fixed point. Moreover, there is a map $f_{\text{fix}} \in [D \to D]$ such that $f_{\text{fix}}(f)$ is, for $f \in [D \to D]$, the least fixed point of $f$.

Proof. Define $f_{\text{fix}}(f) = \text{sup}_D (f^n(\perp): n \in \mathbb{N})$, where $\perp (= \text{sup}_D \emptyset)$ is the least element of $D$ and $f^n(d) = d, f^{n+1}(d) = f(f^n(d)), \forall n \in \mathbb{N}$. "

This has been pointed out by G. Longo, in conversation.
Let $\text{id}_D$ be the identity on $D$. So, e.g., $\text{id}_{[D \to D]}(f) = f$, $\forall f \in [D \to D]$.

1.3. **Definition.** (i) A complete lattice $D$ is (a) reflexive (domain) if $[D \to D]$ is a retract of $D$; i.e., there are continuous maps $F: D \to [D \to D]$ and $G: [D \to D] \to D$ such that $F \circ G = \text{id}_{[D \to D]}$. $(F, G)$ is a retraction pair with retraction maps $F$ and $G$.

(ii) If, moreover, $G \circ F = \text{id}_D$, then $D$ is an additive domain, while, if $G \circ F = \text{id}_D$, then $D$ is an extensional domain.

1.4. **Examples.** (i) A well-known additive domain is the Graph Model $P\omega = \{x: x \subseteq \mathbb{N}\}$, partially ordered by set inclusion (see [24]). To define the retraction maps, let

$$(n, m) = \frac{1}{2}(n + m)(n + m + 1) + m$$

be the Cantor coding of natural numbers and $(e_n)_{n \in \mathbb{N}}$ be an effective enumeration of the finite subsets of $\mathbb{N}$ via $e_n = \{k_0, k_1, \ldots, k_{m-1}\}$, with $k_0 < k_1 < \cdots < k_{m-1}$, iff

$$n = \sum_{i \leq n} 2^{k_i}.$$ 

Then

$$F(x)(y) = \{m: \exists e_n \subseteq y \ (n, m) \in x\}$$

and

$$G(f) = \{(n, m): m \in f(e_n)\}.$$ 

(ii) Scott’s inverse limit construction $D_{\omega}$ (see [22] or [23]) is an extensional domain. Somewhat easier, extensional domains can be “derived” from $P\omega$ [24, 25, 13] (but see [12] for a general construction).

From now on, let $D$ be an arbitrarily fixed additive domain.

1.5. **Definition.** (i) The set of $\lambda$-terms over $D$ (notation: $A(D)$) is defined inductively by

$$x_0, x_1, \ldots \in A(D),$$

(variables)

$$d \in D \Rightarrow c_d \in A(D),$$

(constants over $D$)

$$M, N \in A(D) \Rightarrow (MN) \in A(D),$$

$$M \in A(D) \Rightarrow (\lambda x \cdot M) \in A(D).$$


(ii) The *theory X* consists of equations between \( \lambda \)-terms, axiomatized by the axiom scheme

\[
(\lambda \cdot M) \equiv M_{[x \cdot N]}^{[x \cdot N]}
\]

(where \( M_{[x \cdot N]} \) denotes substitution) and the usual equality axioms and rules, including

\[
M = N \Rightarrow \lambda \cdot M = \lambda \cdot N.
\]

See [2] for the syntactic care needed to define substitution and to insure \( \lambda \)-term-disambiguation.

1.6. **Definition.** Let \( \rho: \text{Variables} \to D \) be a valuation (in \( D \)). One defines, by induction on the structure of \( M \), the *value of \( M \) at \( \rho \) in \( D \) (notation: \( [M]_{\rho}^{D} \)), as follows:

\[
[x]_{\rho}^{D} = \rho(x),
\]

\[
[\text{c}]_{\rho}^{D} = d,
\]

\[
[MN]_{\rho}^{D} = F([M]_{\rho}^{D})([N]_{\rho}^{D})
\]

\[
[\lambda \cdot M]_{\rho}^{D} = G(\lambda x \cdot [M]_{(x = d)}^{D}),
\]

where

\[
\rho(x := d)(y) = \begin{cases} 
\rho(y), & \text{if } y \neq x \\
 d, & \text{if } y = x.
\end{cases}
\]

1.7. **Proposition.** \( [M]_{\rho}^{D} \) is well defined and determines a model of \( \lambda \):

\[
\lambda \vdash M = N \Rightarrow [M]_{\rho}^{D} = [N]_{\rho}^{D}, \quad \text{for all } \rho.
\]

**Proof.** See [3].

1.8. **Notation.**

(i) \( D \models M = N \) iff \( [M]_{\rho}^{D} = [N]_{\rho}^{D} \), for all \( \rho \). This notion is extended in the obvious way to first-order formulas.

(ii) We loosely use, e.g., \( \lambda x \cdot x \cdot d \) to denote \( [\lambda x \cdot x \cdot d]_{\rho}^{D} \in D \), or \( \lambda x \cdot f(x) \) to denote \( G(f) \), for continuous \( f: D \to D \).

Note that additivity of \( D \) implies \( D \models x \in \lambda y \cdot xy \). Moreover, the finitary sup-operation is continuous.
1.9. **Definition.** Let \( a \in D \). Then, where \( f \circ g := \lambda x \cdot f(gx) \), we say that

(i) \( a \) is a retract if \( a = a \circ a \),
(ii) \( a \) is a closure if \( a \) is a retract and \( \mathcal{S} := \lambda x \cdot x \in a \).
(iii) \( a^* = \{ ax : x \in D \} \). Notation: \( x \in a \) iff \( x \in a^* \).

A retract \( a \) (= \( \lambda x \cdot ax \)) is, in fact, a retraction map from \( D \) onto \( a^* \).

The following construction, due to P. Martin-Löf, P. Hancock, and D. Scott, independently, shows that the set

\[ \{ a \in D : a \text{ is a closure} \} \]

is itself of the form \( \mathcal{V}^* \), for some closure \( \mathcal{V} \in D \).

As the map \( f(x, y) = x \cup y \) is continuous, the following makes sense.

1.10. **Definition.** \( \mathcal{V} := \lambda xy \cdot f_{xy} (\lambda z \cdot y \cup xz) \).

1.11. **Lemma.** For all \( x, y \in D \),

(i) \( \mathcal{V} \cdot x(\mathcal{V} \cdot xy) = \mathcal{V} \cdot xy \) and
(ii) \( \mathcal{V} \cdot x \) is a closure.

**Proof.** (i) Note that

\[ \mathcal{V} \cdot xy = y \cup x(\mathcal{V} \cdot xy). \]  

Hence

\[ y \in \mathcal{V} \cdot xy \text{, } \]  

\[ x(\mathcal{V} \cdot xy) \subseteq \mathcal{V} \cdot xy. \]

But \( \mathcal{V} \cdot x(\mathcal{V} \cdot xy) \) is the least \( z \) such that

\[ z = \mathcal{V} \cdot xy \cup xz. \]  

Now \( z = \mathcal{V} \cdot xy \) satisfies (4) by (3); moreover \( z = \mathcal{V} \cdot xy \cup xz \Rightarrow \mathcal{V} \cdot xy \subseteq z \). So we have (i).

(ii) By (i) and (2).

1.12. **Theorem.** (i) For all \( x \in D \), \( x \) is a closure iff \( x \in \mathcal{V}^* \).

(ii) \( \mathcal{V}^* \) is a closure (that is: \( \mathcal{V} \subseteq \mathcal{V}^* \)).

**Proof.** Use Lemma 1.11, noting that if \( x \) is a retract then \( x = \lambda y \cdot xy \), \n\( \mathcal{V} = \lambda x \cdot \mathcal{V} \cdot x \) (since \( \mathcal{V} \) is an abstract) and \( D \) is additive.

As usual, let \( \mathcal{F} := \lambda xy \cdot x \in D \). Then, for \( a \in D \), \( \mathcal{F} \cdot a = \lambda x \cdot a \).
1.13. Definition. (i) \( \mathcal{G} := \lambda x y z v (u v)(x(v)) \).
(ii) \( \lambda x : a \cdot b := (\lambda x : b) \cdot a \) (i.e., \( = \lambda x : b \cdot (ax) \)).
(iii) \( \pi x : a \cdot b := G a(\lambda x : b) \) (i.e., \( = \lambda x y : b(\lambda x : b(x(y))) \)).
(iv) \( a \rightarrow b := G a(\pi b) \) (i.e., \( = \lambda x : b \cdot x \cdot a \)).

1.14. Proposition. Let \( \mathcal{G} \in \mathcal{H} \). Then

(i) \( \forall x \in a \cdot b[x] \in \mathcal{H} \Rightarrow (\pi x : a \cdot b[x] \in \mathcal{H} \).
(ii) \( \forall x \in a \cdot b[x] \in \mathcal{H} \Rightarrow (\lambda x : a \cdot b[x] \in \mathcal{H} \).
(iii) \( \forall x \in a \cdot b[x] \Rightarrow (\lambda x : a \cdot b[x] \in \mathcal{H} \).
(iv) \( \forall x \in a \cdot b[x] \Rightarrow (\lambda x : a \cdot b[x] \in \mathcal{H} \).

Proof. (i) \( (\Rightarrow) \): Assume

\( \forall x \in a \cdot b[x] \in \mathcal{H} \).
write for convenience,

\( A := \pi x : a \cdot b[x] \).

We show \( A \) is a closure. Indeed, writing \( b[a] \) for \( b[x] \cdot a \), etc., one has

\[
A(Ax) = \lambda y : b[ay](b[aoay])(x(ay)) = \lambda y : b[ay](b[aoay])(x(ay)) = \lambda y : b[ay](x(ay)) = Ax.
\]

Moreover,

\( x \in \lambda y : x y \) since \( D \) is additive,
\( \in \lambda y : x(ay) \) since \( a \in \mathcal{H} \),
\( \in \lambda y : b[ay](x(ay)) \) since \( b[ay] \in \mathcal{H} \),
\( \in Ax \).

(\( \Leftarrow \)): Assume

\( \pi x : a \cdot b[x] \in \mathcal{H} \).

Then, with \( A \) as above, one has \( A(Ax) = Ax \) and \( x \in Ax \) (\( \forall x \in D \)). Hence

\( b[ay](b[oy](x(ay)) = b[oy](x(ay)) \).
So
\[ b_{[\omega]} (b_{[\alpha\beta]} z) = b_{[\alpha\beta]} z, \quad \text{(take } x = \lambda w \cdot z), \]
and therefore
\[ \forall x \in a \ b_{[x]} \text{ is a retract.} \quad (1) \]
Moreover,
\[ x \subseteq \lambda y \cdot b_{[\omega]} (x(ay)). \]
So
\[ xy \subseteq b_{[\alpha\beta]} (x(ay)), \]
wherefrom, with \( x = \lambda w \cdot z, \)
\[ z \subseteq b_{[\alpha\beta]} z \]
and therefore
\[ \forall y \in a \ y \subseteq b_{[y]}. \quad (2) \]
By (1) and (2),
\[ \forall y \in a \ b_{[y]} \text{ is a closure.} \]
Hence Theorem 1.12(i) applies.
(ii) \( \Rightarrow \): Assume
\[ \forall x \in a \ b_{[x]} \in c_{[x]}. \]
Then
\[
(\pi x: a \cdot c_{[x]})(\lambda x: a \cdot b_{[x]} = \forall \alpha (\lambda x: a \cdot c_{[x]})(\lambda x: b_{[x]} \circ a)
\]
\[ = \lambda y \cdot c_{[\alpha\beta]}(b_{[\alpha\beta]})) \]
\[ = \lambda y \cdot c_{[\alpha\beta]}(b_{[\alpha\beta]}) \quad \text{since } a \notin \mathcal{F}, \]
\[ = (\lambda x: b_{[x]} \circ a \quad \text{by assumption}, \]
\[ = \lambda x: a \cdot b_{[x]} \]
Therefore
\[ (\lambda x: a \cdot b_{[x]}) \in (\pi x: a \cdot c_{[x]}). \]
(⇐): Assume

\((\lambda x: a \cdot b_{|x|}) \in (\pi x: a \cdot c_{|x|})\).

Then

\[ \lambda x \cdot b_{|ax|} = (\pi x: a \cdot c_{|x|})(\lambda x: a \cdot b_{|x|}) \]
\[ = \lambda y \cdot c_{|ay|}(b_{|ay|}). \]

Hence

\[ b_{|ax|} = c_{|ax|}(b_{|ax|}), \]
i.e.,

\[ \forall x \in a \ b_{|x|} \in c_{|x|}. \]

(iii) First note that

\[ f = (\pi x: a \cdot b_{|x|})f \Rightarrow f = (\pi x: a_{|x|})(\lambda x: a \cdot b_{|x|})f \]
\[ = b_{|ax|}(f(ax)). \]
So

\[ f \in (\pi x: a \cdot b_{|x|}) \Rightarrow \forall x \in a f_{|x|} = b_{|x|}(f_{|x|}) \]
\[ \Rightarrow \forall x \in a f_{|x|} \in b_{|x|}. \]

Now

\[ f \in (\pi x: a \cdot b_{|x|}) \Rightarrow f = (\pi x: a_{|x|})(\lambda x: a \cdot b_{|x|})f \]
\[ = \lambda x \cdot b_{|ax|}(f(ax)) \]
\[ = \lambda x \cdot f(ax), \quad \text{by the above}, \]
\[ = \lambda x: a \cdot f_{|x|}. \]

(iv) Immediate, from (ii) and (iii). □

1.15. Remark. Let \( a \) be a closure. Define

\[ \perp_a := a \perp \quad \text{and} \quad \mathcal{Y}_a := f \circ (a \circ - a). \]

Then

(i) \( a^* \) is an algebraic lattice.
(ii) \( \perp_a \) is the least element of \( a^* \);
(iii) \( \mathcal{Y}_a (a \circ - a) \circ - a. \)
(iv) For all \( x \in D \), if \( x \in (a \circ - a) \) then \( \mathcal{Y}_a x \) is the least fixed point of \( x \) in \( a^* \).
Proof. (i) See [7].

(ii) In $D$ one has
\[ \forall x \bot \subseteq x; \]
hence, by monotonicity,
\[ \forall x a \bot \subseteq ax, \]
i.e.,
\[ \forall x \in a^\bot \subseteq x. \]

(iii) Easy computations show
\[ \mathcal{Y}_a \circ (a \rightarrow a) = \mathcal{Y}_a \]
and
\[ a \circ \mathcal{Y}_a = \mathcal{Y}_a. \]
Therefore
\[ a \circ \mathcal{Y}_a \circ (a \rightarrow a) = \mathcal{Y}_a \]
and we are done.

(iv) Let $x \in (a \rightarrow a)$. Then
\[ \mathcal{Y}_a x = fix((a \rightarrow a) x) = fix x = x(fix x) = x(fix((a \rightarrow a) x) = x(\mathcal{Y}_a x). \]
Moreover, if $xy = y$ then
\[ \mathcal{Y}_x x = fix x \subseteq y. \]

2. CLASSICAL AUTOMATH: Syntax and Semantics

In first-order logic one first defines two recursive syntactic categories: terms and well-formed formulas (wffs); after that one can define the set of provable formulas as a subset of wffs.

In Classical AUTOMATH (CA) the situation is similar, but more complex (viz., roughly comparable with that encountered in Martin-Löf's type theories [16, 17]).
First, one defines (this is a "correctness-free" description stage) the following recursive syntactic categories:

- terms,
- sentences (E- resp. Q-sentences),
- contexts,
- lines (primitive and defining lines), while a book is a finite set of lines.

Then one defines (the "correctness" description stage) the r.e. set of provable formulas (or statements) of CA of the form

\[ \Delta \vdash_B \phi \]

(where B is a book, \( \Delta \) is a context and \( \phi \) is an E- or a Q-sentence of CA; accurately, \( \vdash \), the classical de Bruijn type-assignment, is a ternary relation and \( \Delta \vdash_B \phi \) is shorthand for \( \langle B, \Delta, \phi \rangle \in \vdash \)).

The intuition behind this is as follows. If \( a, b \) are terms then

\[ a : b \]

is an E-sentence, with the intended meaning "\( a \) is of type \( b \)." So terms denote (as in [16, 17]) both types and objects. Moreover,

\[ a = b \]

is a Q-sentence and can be read as "\( a \) is convertible to \( b \)" or as "\( a \) is definitionally equal to \( b \)." A context is a finite sequence (not just a finite set, not a Curry basis, say; cf. [9])

\[ \Delta : = \varphi_1, \ldots, \varphi_n \]

of E-sentences \( \varphi_i : = v_i : a_i \), where the \( \vec{\varphi} : = v_1, \ldots, v_n \) are pairwise distinct variables and \( \vec{\alpha} : = a_1, \ldots, a_n \) are terms. Each \( \varphi_i \) of this form is called an assumption in \( \Delta \) and each assumption \( \varphi_i \) "declares" a variable \( v_i \) of type \( a_i \).

The terms are formed from variables, a "universe constant" \( \tau \) (denoting the type of all types), closed under application, typed abstraction, cartesian products and explicit function-definition (so, if for \( v \) of type \( a \), the term \( b[v] \) is of type \( b[i] \) then \( \lambda v : a \cdot b[v] \) is a function of type \( \tau v : a \cdot b[i] \), while if \( c \) is an n-ary function constant and \( \vec{\alpha} : = a_1, \ldots, a_n \) are terms of appropriate types then \( c(\vec{\alpha}) = c(a_1, \ldots, a_n) \) is the value of a function having as type the appropriate "generalized" cartesian product.

The lines of CA (also called "constructions" in [21]) serve to specify the behaviour of function constants in CA. They are of the form

\[ \{ c = a : b \}, \]
where $c$ is either a primitive or a defined constant, $a$ is the definiens of $c$ and $b$ is its type. In general, if $c$ is an $n$-ary defined constant its definiens is of the form $a := \lambda \nu : f \cdot a'$, whereas, if $c$ is primitive, one would want to specify its pseudo-definiens (just to signal that $c$ is primitive) as $a := \lambda \nu : f \cdot c(\nu)$. Correspondingly, the type of an $n$-ary constant $c$ is of the form $b := \pi \nu : f \cdot b'$.

Terms, $E$-sentences, $Q$-sentences and contexts that occur in some provable statement

$$A \vdash \varphi$$

of CA (but not in $B$) are called (CA-) correct, while a book $B$ is (CA-) correct if (*) is provable for some context $A$ and some sentence $\varphi$.

In fact, CA-correct books (also called "compatible sites" in [21]) play exclusively the role of a book-keeping device and are used to "store" information concerning the behaviour of the function constants in CA (cf. with the "theories" in first-order logic).

The formal description of the CA-syntax is now as follows.

2.1. DEFINITION (CA "correctness-free" syntax). (i) The alphabet of CA consists of

1° a set $\text{Var} = \{e_i : i \in \mathbb{N}\}$ of variables;

2° for each $n \in \mathbb{N}$, sets

$$\text{Pcons}_n = \{p_i^n : i \in \mathbb{N}\} \text{ and } \text{Dcons}_n = \{d_i^n : i \in \mathbb{N}\}$$

of primitive resp. defined constants of arity $n$ (p- resp. d-constants, for short);

3° a "universe symbol": $\tau$;

4° abstractors: $\lambda, \Pi$;

5° (binary) predicates: ("... has type ..."),

$$= ("... equals ...");$$

6° auxiliary symbols ("punctuation"): $:, ( ) \leftrightarrow \{ \}$.

Syntactic Variables

$v, v', v''...$ range over $\text{Var}$,

$p, p',...$ range over $p$-constants,

d, d',... range over $d$-constants,

c, c',... range over Cons

$\text{Cons} = \bigcup_{n \in \mathbb{N}} (\text{Pcons}_n \cup \text{Dcons}_n)$. 

(ii) The set Term of (CA-)terms is defined inductively by
1° \( \text{Var} \subseteq \text{Term}, \tau \in \text{Term} \).
2° If \( c \in \text{Pcons}_n \cup \text{Dcons}_n \ (n \in \mathbb{N}) \) and \( a_1, \ldots, a_n \in \text{Term} \) then \( c(a_1, \ldots, a_n) \in \text{Term} \).
3° If \( a, b \in \text{Term} \) then \( (ab), (\lambda v : a \cdot b), (\Pi v : a \cdot b) \in \text{Term} \).

Syntactic Variables
\( a, b, \ldots, f, g, h, \ldots \) (with sub- and/or superscripts) range over Term.

(iii) The sentences of CA, ranged over by \( \phi, \phi' \ldots \), are
1° \( \forall \)-sentences, of the form \( a : b \),
2° \( \forall \)-sentences, of the form \( a = b \).

(iv) A (CA-)context is a sequence \([v_1 : a_1, \ldots, v_n : a_n]\), where \( v_1 : a_1, \ldots, v_n : a_n \) are \( \forall \)-sentences with the \( v_1, \ldots, v_n \) pairwise distinct. In particular, \([\ ]\) denotes the empty context.

Syntactic Variables
\( \Delta, \Delta', \ldots \) range over contexts.

Notation
If \( \Delta := [v_1 : a_1, \ldots, v_n : a_n] \) then we set
\( \lambda \Delta \cdot b := \lambda v_1 : a_1 \cdots \lambda v_n : a_n \cdot b \) and
\( \Pi \Delta \cdot v := \Pi v_1 : a_1 \cdots \Pi v_n : a_n \cdot b \) resp.

(v) Let \( \Delta := [v_1 : a_1, \ldots, v_n : a_n], n \in \mathbb{N} \) and \( g, h \in \text{Term} \).
1° If \( p \in \text{Pcons}_n \) then \( \{p = \lambda \Delta \cdot p(\delta) : \Pi \Delta \cdot h\} \) is a \( \lambda \)-line ("primitive line") in CA.
2° If \( d \in \text{Dcons}_n \) then \( \{d = \lambda \Delta \cdot g : \Pi \Delta \cdot h\} \) is a \( d \)-line ("defining line") in CA.
3° A (CA-)line is either a \( \lambda \)-line or a \( d \)-line in CA.

(vi) A (CA-)book is a finite set of CA-lines.

Syntactic Variables
\( B, B', \ldots \) range over books.

2.2. Notation. Conventions. (i) Terms are identified modulo uniform reletterings of their bound variables (where the bound/free variables in a term are defined in the usual way; note, however, that one has here two distinct abstractors: \( \lambda \) and \( \Pi \)).
(ii) Where $\vec{v} := v_1, \ldots, v_n$ (with the $v_i$'s pairwise distinct) and $b, \vec{a} := a_1, \ldots, a_n$ are terms, the notation

$$b|_{\vec{v}} = \vec{a}$$

stands for simultaneous substitution (for $n = 1$, this becomes usual substitution).

(iii) For $\vec{a}$ as above and $c$ an $n$-ary function constant we write

$$c(\vec{a}) := c(a_1, \ldots, a_n),$$

while, if $\Delta := [v_1 : a_1, \ldots, v_n : a_n]$ and $a$ is a term, then

$$\Delta[v : a] := [v_1 : a_1, \ldots, v_n : a_n, v : a].$$

(iv) Finally, in the next definition,

1° $\Delta \vdash_B a$ is shorthand for $\Delta \vdash_B a = a$,

2° $\Delta \vdash_B a, b$ is shorthand for $(\Delta \vdash_B a) \& (\Delta \vdash_B b)$, and

3° $\Delta \vdash_B f : g : h$ stands for $(\Delta \vdash_B f : g) \& (\Delta \vdash_B g : h)$.

2.3. Definition. (i) A statement of CA is of the form

$$\Delta \vdash_B \varnothing$$

where $B$ is a book, $\Delta$ is a context and $\varnothing$ is an ($E$- or $Q$-) sentence.

(ii) The provable statements of CA are inductively defined by the following set of (correctness) rules.

Correctness Rules of CA

1° Structural rules. Let $\Delta := [v_1 : f_1, \ldots, v_n : f_n]$.

1°1 Initialization:

$$[ ] \vdash_{\psi} \tau.$$  \hfill (r$_{\tau}$)

1°2 Book-recursion ($n \geq 0$):

$$\Delta \vdash_{B} \tau \Rightarrow [ ] \vdash_{B, \tau},$$  \hfill (p1)

where

1. $p \in \text{Pcons}_n$, fresh for $\vec{f}, B$,

2. $B' = B \cup \{p = \lambda \bar{a}. p(\vec{b}) : \Pi \bar{a} \cdot \tau\}$,

3. $\Delta \vdash_{B} g : \tau \Rightarrow [ ] \vdash_{B, \tau}$.  \hfill (p2)
where

\[ p \in P_{\text{cons}}, \text{fresh for } f, g, B, \]
\[ B' = B \cup \{ p = \lambda d \cdot p(\overline{\nu}) : \Pi d \cdot g \} \].

\[ \Delta \vdash_B g : h \Rightarrow \Delta[\nu : \tau] \vdash_B \tau \quad (d) \]

where

\[ d \in D_{\text{cons}}, \text{fresh for } f, g, h, B, \]
\[ B' = B \cup \{ d = \lambda d \cdot g : \Pi d \cdot h \} \].

1.3 Context-recursion \((n \geq 0)\):

\[ \Delta \vdash_B \tau \Rightarrow \Delta[\nu : \tau] \vdash_B \tau \quad (c1) \]

provided \( \nu \) is fresh for \( \Delta \).

\[ \Delta \vdash_B g : \tau \Rightarrow \Delta[\nu : g] \vdash_B \tau \quad (c2) \]

provided \( \nu \) is fresh for \( \Delta, g \).

1.4 Projection rules:

1.41 Book-projection \((n \geq 0)\):

\[ \Delta \vdash_B \tau; \Delta' \vdash_B \tau \]
\[ \Delta' \vdash_B \alpha_i : f_{i\in \{1\ldots n\}} (1 \leq i \leq n) \Rightarrow \Delta' \vdash_B \xi(\overline{\alpha}) = g_{|\overline{\alpha} = \vec{a}} \quad (bp) \]
\[ \xi = \lambda d \cdot g : \Pi d \cdot h \in B. \]

1.42 Context-projection \((n \geq 1)\):

\[ \Delta \vdash_B \tau \Rightarrow \Delta \vdash_B \nu_i : f_i \quad (1 \leq i \leq n). \quad (cp) \]

1.43 \(E\)-sentence-projection:

\[ \Delta \vdash_B \alpha : b \Rightarrow \Delta \vdash_B \alpha \quad (r_1) \]
\[ \Delta \vdash_B \alpha : b \Rightarrow \Delta \vdash_B b. \quad (r_2) \]

1.5 Substitution \((n \geq 0)\):

\[ \Delta \vdash_B \tau; \Delta' \vdash_B \tau \]
\[ \Delta' \vdash_B \alpha_i : f_{i\in \{1\ldots n\}} (1 \leq i \leq n) \Rightarrow \Delta' \vdash_B \xi(\overline{\alpha}) = h_{|\overline{\alpha} = \vec{a}} \quad (sub) \]
\[ \xi = \lambda d \cdot g : \Pi d \cdot h \in B. \]
2° Assignment rules:

\[ \Delta \vdash_g g : \tau \]
\[ \Delta[v : g] \vdash_h h_{[v]} : \tau \] \quad \Rightarrow \quad \Delta \vdash_{g \cdot h_{[v]}} \tau. \quad (III) \]

\[ \Delta \vdash_a a : g : \tau \]
\[ \Delta \vdash_{g \cdot h_{[v]}} \tau \] \quad \Rightarrow \quad \Delta \vdash_{h_{[\Delta = a]}} \tau. \quad (IIE) \]

\[ \Delta \vdash_{\lambda v : g} \tau \]
\[ \Delta[v : g] \vdash_{h_{[v]}} \tau \] \quad \Rightarrow \quad \Delta \vdash_{\lambda v : g \cdot f_{[v]} : h_{[v]}} \tau. \quad (abs) \]

\[ \Delta \vdash_a a : g : \tau \]
\[ \Delta \vdash_{g \cdot h_{[v]}} \tau \] \quad \Rightarrow \quad \Delta \vdash_{f a : h_{[\Delta = a]}}. \quad (app) \]

3° Conversion rules:

3°1 Equivalence:

\[ \Delta \vdash_a a = b \Rightarrow \Delta \vdash b = a \] \quad (s) \]
\[ \Delta \vdash_f f = g ; \Delta \vdash g = h \Rightarrow \Delta \vdash_f = h. \] \quad (t) \]

3°2 Congruence:

\[ \Delta \vdash_a a = b ; \Delta \vdash_{\Phi_{[\Delta =]}}, \Phi_{[\Delta =]} \Rightarrow \Delta \vdash_{\Theta_{[\Delta =]}} \Phi_{[\Delta =]} = \Phi_{[\Delta =]} \] \quad (mon) \]

where \( \Phi_{[\Delta =]} \) is of one of the forms

\[ v g, f v, \lambda v : g \cdot h, \Pi v : g \cdot h, c(a_1, \ldots, v, \ldots, a_n) \]

with \( v \) occurring in \( \Phi_{[\Delta =]} \) only at the indicated place (just once) and \( \Phi_{[\Delta =]} := \Phi_{[\Delta =]} \) etc.

\[ \Delta \vdash_g g : \tau \]
\[ \Delta[v : g] \vdash_a b \] \quad \Rightarrow \quad \Delta \vdash_{g \cdot a = \lambda v : g \cdot b}. \quad (\xi_a) \]

\[ \Delta \vdash_g g : \tau \]
\[ \Delta[v : g] \vdash_a b \] \quad \Rightarrow \quad \Delta \vdash_{g \Delta = a \cdot g \cdot b}. \quad (\xi_\Pi) \]

\[ \Delta \vdash a_1 : b ; \Delta \vdash a_1 = a_2 \Rightarrow \Delta \vdash_{a_2 : b}. \] \quad (eqs) \]

\[ \Delta \vdash a : b_1 ; \Delta \vdash b_1 = b_2 \Rightarrow \Delta \vdash a : b_2. \] \quad (eqp) \]

3°3 Evaluation:

\[ \Delta \vdash_a a : g : \tau \]
\[ \Delta[v : g] \vdash_{f_{[v]} : h_{[v]}} \] \quad \Rightarrow \quad \Delta \vdash_{(\lambda v : g \cdot f_{[v]} = f_{[\Delta = a]}) a} \quad (\beta) \]
3.4 Functionality:

\[ \Delta \vdash \beta \, f' : (hv : g \cdot h_{(v)}) \vdash \Delta \vdash \beta \lambda v : g \cdot fv = f \]

provided \( v \) is not free in \( f \).

Now one can give the semantics of CA (in any additive domain \( D \)).

2.4. Definition. (i) A \( v \)-valuation in \( D \) is a map \( p : \text{Var} \rightarrow D \).

(ii) A \( c \)-valuation in \( D \) is a map \( \xi : \text{Cons} \rightarrow D \).

(iii) The value (= interpretation) if a (CA-) term \( f \) relative to \( p \) and \( \xi \) in \( D \) (notation: \( \llbracket f \rrbracket^D_{p,\xi} \)) is defined by induction on the structure of \( f \) as follows:

1° \[ \llbracket v \rrbracket^D_{p,\xi} = p(v) \]

2° \[ \llbracket c \rrbracket^D_{p,\xi} = \xi(c) \]

3° \[ \llbracket \alpha \rrbracket^D_{p,\xi} = \mathcal{F} \]

4° \[ \llbracket ab \rrbracket^D_{p,\xi} = F(\llbracket a \rrbracket^D_{p,\xi})(\llbracket b \rrbracket^D_{p,\xi}) \]

5° \[ \llbracket \lambda v : a \cdot b \rrbracket^D_{p,\xi} = (G(\lambda d : \llbracket b \rrbracket^D_{p(\alpha(v)=d),\xi})) \circ (\llbracket a \rrbracket^D_{p,\xi}) \]

6° \[ \llbracket hv : a \cdot b \rrbracket^D_{p,\xi} = \mathcal{G}(G(\lambda d : \llbracket b \rrbracket^D_{p(\alpha(v)=d),\xi}))\llbracket a \rrbracket^D_{p,\xi} \]

(iv) An \( E \)-sentence \( \varphi : a : b \) is true at \( p, \xi \) in \( D \) (notation: \( D, p, \xi \models a : b \)) if

\[ \llbracket a \rrbracket^D_{p,\xi} \in \llbracket b \rrbracket^D_{p,\xi} \]

Similarly, a \( Q \)-sentence \( \varphi' := a = b \) is true at \( p, \xi \) in \( D \) (notation: \( D, p, \xi \models a = b \)) if

\[ \llbracket a \rrbracket^D_{p,\xi} = \llbracket b \rrbracket^D_{p,\xi} \]

(v) For all contexts \( \Delta := [v_1 : f_1, ..., v_n : f_n] \) in CA, all \( E \)-sentences \( a : b \) and all \( Q \)-sentences \( a = b \), one defines

\[ D, p, \xi \models \Delta \vdash a : b \Leftrightarrow [\lambda \Delta : a]_{p,\xi} \in [\Pi \Delta : b]_{p,\xi} \]

resp.

\[ D, p, \xi \models \Delta \vdash a = b \Leftrightarrow [\lambda \Delta : a]_{p,\xi} = [\lambda \Delta : b]_{p,\xi} \]

(vi) Let \( B \) be a CA-book such that, for \( 0 \leq i \leq n, 0 \leq j \leq m \),

\[ \{ p_i = \lambda d : p_i(\bar{v}) : \Pi d : f_i \} \]

are the \( p \)-lines of \( B \)

and

\[ \{ d_j = \lambda d : g_j : \Pi d : h_j \} \]

are the \( d \)-lines of \( B \).
Then \( B \) is satisfied at \( \rho, \xi \) in \( D \) (notation: \( D, \rho, \xi \models B \)) if

1° \( \forall i: 0 \leq i \leq n \Rightarrow (D, \rho, \xi \models p_i : \Pi \Delta \cdot f_i) \)

2° \( \forall j: 0 \leq j \leq m \Rightarrow (D, \rho, \xi \models d_j \triangleq \lambda \Delta \cdot g_j) \& (D, \rho, \xi \models d_j : \Pi \Delta \cdot h_j) \).

(vii) Finally, one defines validity for CA-statements by

\[
D \models^A a : b \iff (D, \rho, \xi \models B \Rightarrow D, \rho, \xi \models^A a : b)
\]

resp.

\[
D \models^A a = b \iff (D, \rho, \xi \models B \Rightarrow D, \rho, \xi \models a = b).
\]

The main result can be now stated as follows.

2.5. Theorem (Soundness for CA). Let \( D \) be an arbitrary additive domain. Then for all CA-books \( B \), all CA-contexts \( \Delta \) and all CA-terms \( a, b \),

(i) \( \Delta \vdash^A a : b \Rightarrow D \models^A a : b \),

(ii) \( \Delta \vdash^A a = b \Rightarrow D \models^A a = b \).

Proof: Induction on the generation of \( \Delta \vdash^A a : b \) and \( \Delta \vdash^A a = b \), in CA, using Proposition 1.14.

3. Discussion: Related Systems

As described in Definitions 2.1 and 2.3, the syntax of CA diverges unessentially from standard presentations of AUT-68 (= the “reference” version of CA; cf. [6, 21], etc.).

In fact, the main differences from [21], say, are notational in nature and are justified by model-theoretic as well as readability considerations: abstraction terms, denoted by \([v : a]\) in AUT-68, are disambiguated here according to their intended meaning, application terms \((fa)\) have function-part on the l.h.s. (as it is usual in lambda-calculus), while the official AUT-68 lines have here somewhat a more “portable” format, fitting straightforwardly the interpretation in Definition 2.4. Finally, in the “reference” AUT-68 version, a book is a sequence of lines; so the present concept would rather correspond to the “sites” of [21].

As regards CA-correctness (Definition 2.3), the main novelty over [21].2 consists of the elimination of the statements

\( \Delta \vdash^A a \).
from the primitive CA-syntax. So the reflexivity rule

\[ \Gamma \vdash_B a \Rightarrow \Delta \vdash_B a = a \]  \hspace{1cm} (r)

follows trivially here, by mere notational conventions. In particular, the hypotheses of \((\eta)\) are, obviously, necessary in the present setting. Finally, (eqs) is, apparently, derivable from the remaining correctness rules of CA (as in [6, 21]), but \((r_p)\) seems necessary, due to the fact that "\(\Delta \vdash_B a\)" is a defined notion.

Let CA\(_0\) be the "pure" part of CA; i.e., CA without function constants (so CA\(_0\) has no lines and books). Then a possible set of primitive correctness rules for CA\(_0\) is (cf. Definition 2.3 above)

1° \((r), (c1), (c2), (cp), (r_p), (sub),\)

2° \((\Pi), (\Pi e), (abs), (app),\)

3° \((s), (t), (mon), (\xi^n), (\xi^n), (eqs), (eqp), (\beta), (\eta),\)

where \(B = \emptyset\) (or just omitted) and \((sub)\) is the following analogue of \((sub)\): for \(\Delta : = \{v_1 : f_1, \ldots, v_n : a_n\}, n \geq 1,\)

\[ A \vdash g : h \]

\[ A' \vdash \tau \]

\[ A'' \vdash a_i : f_i[v_i = a] \quad (1 \leq i \leq n) \]

This system is very useful, for most of the known typed lambda-calculi can be obtained from it, by trivial modifications in the set of its correctness rules.

First note that \((sub)\) is derivable in CA (cf. [6, 21]), so CA\(_0\) is actually a subsystem of CA. For a mild combinatory variant of CA\(_0\), see, e.g., [26].

Consider now the following axioms:

\[ [ \ ] \vdash \tau : \tau \]  \hspace{1cm} (r)

\[ [ \ ] \vdash \tau : (\tau \rightarrow \tau) \]  \hspace{1cm} (\tau\tau)

where \(a \rightarrow b : = \Pi v : a \cdot b, \) provided \(v\) is not free in \(b.\) Both \((r)\) and \((\tau\tau)\) are valid in additive domains: for \((r)\) this follows from Theorem 1.12(ii), whereas, for \((\tau\tau),\) one checks easily that

\[ \forall x : (\forall x \rightarrow \forall x) \forall x \]

in any additive domain \(D.\) This shows that CA\(_0\) is "classically" consistent with any one of \((r), (\tau\tau);\) that is, one cannot derive in the resulting extensions

\[ [ \ ] \vdash a : b \]
for all (closed) terms \( a, b \) (and similarly for \( Q \)-sentences \( a = b \)). For further reference, let \( CA_\infty := CA_0 + (\tau) \) and \( CA_{\tau\tau} := CA_0 + (\tau\tau) \). \( CA_\infty \) (which is, in fact, Martin-Löf’s system in [14]) is known to be “intuitionistically” inconsistent, in the sense that it allows proving

\[
\forall g \in \text{Term} \exists f \in \text{Term} \left[ \vdash g : \tau \Rightarrow \vdash f : \tau \right] \quad \text{(GP)}
\]

(in other words: all closed \( CA_\infty \)-correct types are inhabited). This is the so-called Girard Paradox (cf. [8, 15]) and shows, essentially, that \( CA_\infty \) is inconsistent with the “formulae-as-types”-interpretation [10, 16], preferred in intuitionistic type theory and some version of AUTOMATH (e.g., in AUTQE; see [11]).

Starting from \( CA_0 \) one can obtain easily “AUT-like”-formalizations of the First- and Second-Order Typed Lambda-Calculi \( \lambda^1, \lambda^2 \) (cf. [9] for \( \lambda^1 \) and [8, 20] for \( \lambda^2 \)).

Note first that the following weaker assignment rules (labelled collectively (ass-1), say) are derivable in \( CA_0 \):

\[
\Delta \vdash g : \tau; \Delta \vdash h : \tau \Rightarrow \Delta \vdash g \rightarrow h : \tau \quad \text{(III-1)}
\]

\[
\Delta \vdash \forall \, g : \tau; \Delta \vdash g \rightarrow h : \tau \Rightarrow \Delta \vdash \forall \, h : \tau \quad \text{(IIe-1)}
\]

\[
\Delta \vdash g : \tau; \Delta \vdash \varphi (\alpha) : \tau \Rightarrow \Delta \vdash \lambda \varphi : g \cdot f : g \rightarrow h \quad \text{(abs-1)}
\]

\[
\Delta \vdash a : g : \tau; \Delta \vdash f : g \rightarrow h \Rightarrow \Delta \vdash fa : h. \quad \text{(app)-1)}
\]

Similarly, the rules (ass-1), together with the following rules (labelled, for convenience, (ass-2), say), are easily seen to be derivable in \( CA_\infty \):

\[
\Delta \vdash \varphi (\alpha) : \tau \Rightarrow \Delta \vdash \Pi \varphi : \tau \cdot h : \tau \quad \text{(III-2)}
\]

\[
\Delta \vdash a : \tau; \Delta \vdash \Pi \varphi : \tau \cdot h : \tau \Rightarrow \Delta \vdash h_{[\alpha = \alpha]} : \tau \quad \text{(IIe-2)}
\]

\[
\Delta \vdash \varphi (\alpha) : \tau \Rightarrow \Delta \vdash \lambda \varphi : \tau \cdot f : \Pi \varphi : \tau \cdot h \quad \text{(abs-2)}
\]

\[
\Delta \vdash a : \tau; \Delta \vdash \Pi \varphi : \tau \cdot h \Rightarrow \Delta \vdash fa : h_{[\alpha = \alpha]} \quad \text{(app)-2)}
\]

Let \( LT_1 \), \( LT_2 \), resp., be the systems obtained from \( CA_0 \) by replacing its assignment rules (\( II \), (\( IIe \)), (abs), (app) by (ass-1), (ass-2), resp. So \( LT_1 \) is a subsystem of \( CA_\infty \), while \( LT_2 \) is a subsystem of Martin-Löf’s \( CA_\infty \). One may guess \( LT_1 \), \( LT_2 \) are conservative extensions (under the obvious translation) over \( \lambda^1, \lambda^2 \), resp. (though a formal proof of this may be somewhat involved). For present purposes it is enough to notice that the First-Order Typed Lambda-Calculus \( \lambda^1 \) can be interpreted in \( CA_0 \) (and hence in \( CA_\infty \)), whereas the Second-Order analogue (= the so-called “parametric” typed lambda-calculus) can be interpreted trivially in \( CA_\infty \). That is, both calculi \( LT_1 \) and \( LT_2 \) (and therefore \( \lambda^1 \) and \( \lambda^2 \), resp.) admit of a
completely similar "fixed-point closure semantics" according to Definition 2.4 above.

Remark 1.15(ii)–(iv) shows that this is also the case for the "AUT-like"-formalization of Pure LCF (cf. [18, 19]), i.e., LT₁ extended by fixed-point recursion (and even Scott-induction). Note that the resulting "closure semantics" is different from Milner's [19].

Completeness fails, in additive domains, for all systems named above: indeed (r) is not derivable in CA₀, CAᵥ, LT₁, LT₂, etc., while (rτ) is not derivable in CAᵥ. It is also worthwhile noting that unicity of types

\[ \Delta \vdash a : b₁; \Delta \vdash a : b₁ \Rightarrow \Delta \vdash b₁ = b₂ \quad (\text{UT}) \]

(otherwise derivable in CA₀, CAᵥ, etc.) fails for the above semantics (one checks that \( \emptyset \neq (\emptyset \rightarrow \emptyset) \), in \( \mathcal{P}_0 \), say).

Finally, the present semantics does not work for the AUT-QE system of [11] (this is just QA in [21]; reason: the presence of the rule of Type Inclusion and (app-2) in [21, p. 103]), nor for AUT-SL and LAMBDA-AUTOMATH (cf. [6] for a survey of the syntax), but can be easily extended to Zucker's AUT-Pi [27; 6, Chap. VIII], working directly in \( \mathcal{P}_0 \), say.

Received: December 24, 1983; Accepted: February 29, 1984

References

8. Girard, J.-Y. (1972), "Interpretation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur," Thèse de doctorat d'état, Université de Paris VII.


12. Koyman, C. P. J. (1983), Derived λ-calculus models and the construction of \( D_m \) inside \( F_m \), Preprint No. 283, Department of Mathematics, University of Utrecht. (See also: “Models of the Lambda-Calculus,” Mathematical Centre Tracts, Mathematical Centre, Amsterdam, 1984.)


