Considerations on Automath in Light of the Grundlagen

Ferruccio Guidi

University of Bologna, Italy

ferruccio.guidi@unibo.it

May 26, 2016

1. Overview

- The Automath-related formal systems have a rich set of features, some of which have been largely neglected in subsequent type theory.
- In particular, we want to focus our attention on the next features:

1. the unified binder;
2. the extended applicability condition;
3. the Π-reduction;
4. the weak correctness.

- Landau's Grundlagen formalized in Aut-QE is the foremost product meant to testify the usability and convenience of Automath systems.
- And yet, we do not see in the Grundlagen convincing applications of these features, strongly put forward by the Automath tradition.
- CC has none of them, but accepts an easily translated Grundlagen.

2. The unified binder demystified - Taxonomy

- "Binder" $\Rightarrow$ A typed abstraction $(\flat_x V)$ capable of β-like reductions.

x is the variable on which we abstract, and V is its expected type.

- "Unified" $\Rightarrow$ Unification may occur at three levels:

1. unification in the concrete syntax

($i.e.$, unified binders are disambiguated before entering the kernel);

2. unification in the abstract syntax

($i.e.$, the kernel receives unified binders and disambiguates them);

3. unification in the semantics

($i.e.$, the kernel does not disambiguate the binders).

- The Automath languages use $(\flat_x V)$ to denote several binders with distinct semantics, $i.e.$, $(\lambda_x^\infty V)$ and $(\lambda_x^3 V) : (\lambda_x V) : (\Pi_x V) : (\Pi_x^0 V)$.

- The binder $(\Pi_x^0 V)$ is capable of ζ-like reductions: $(\Pi_x^0 V)\star \to_\upsilon \star$.

3. The unified binder demystified - Applications

- De Bruijn pursues unification in the abstract syntax and semantics.

1. Unification in the abstract syntax, expressive power of $\lambda\!\rightarrow$: OK.

Aut-68: different rules for $(\flat_x V)M$ according to the degree of M.

2. Unification in the semantics, expressive power of $\lambda\!\rightarrow$: OK.

Uniform rules for $(\flat_x V)M$: Aut-QE-NTI, System Λ, $\lambda\lambda$, Λ_∞, $\Delta\Lambda$, λ^λ.

3. Unification in the semantics, more expressive power: KO.

Some desired property is weakened or fails: Aut-QE, $\flat$-Cube.

4. Unification in the abstract syntax, more expressive power: KO?

- We explain in formal terms the KO of choice 3 as follows:

1. with $(\Pi^0_x V)$: $(\lambda_x V) \equiv (\Pi^0_x V) \Rightarrow (N)(\lambda_x V) \equiv (N)(\Pi^0_x V)$

the critical $\beta\upsilon$-pair is not confluent (no Church-Rosser);

2. without $(\Pi^0_x V)$: $(\lambda_x V) \equiv (\Pi_x V) \Rightarrow (\Pi_x V) \equiv \star$ (no unique types).

4. The unified binder demystified - Considerations

1. A slogan: "In Automath, one binder is enough".

- The working systems featuring one binder in the abstract syntax have the power of $\lambda\rightarrow$, which is too low for real large-scale applications.

2. A slogan: "In Automath, $\Pi \equiv \lambda$".

- This is true only in Aut-68, in other cases Π is not present (2. prev. page), $\Pi \not\equiv \lambda$ (Aut-Π), or the systems work badly (3. prev. page).

3. $\Pi \equiv \lambda$ in Aut-QE yields $(\forall_x V)M \equiv (\lambda_x V)M$ in the Grundlagen.

- Identifying a predicate with its universal quantification avoids a handful of $\forall$-introductions at the cost of generating logical confusion.

- The situation is very clear in the line named `all"l"`, where the $\forall$-introduction rule is defined simply as the projection $\sigma, p \mapsto p$.

```
@[sigma:'type'][p:[x:sigma]'prop']all:=p:'prop'
```

5. The extended applicability condition - Example

• The "applicability condition" is the condition on the terms M and N ensuring that M applied to N, displayed $(N)M$, is valid or correct.

1. In a PTS: if $N : V$ and $M : (\Pi_x V)T$, then $(N)M$ is correct.
2. In Aut-QE: if $N : V$ and $M :^n (\flat_x V)T$, then $(N)M$ is correct.

• In the extended applicability (2.), the symbol $:^n$ denotes typing iterated n times, with $0 \leq n < \infty$. If $n = 0$, M reduces to $(\flat_x V)T$.

• The only instance of (2.) with $n \neq 1$ occurs in the next lines of the Grundlagen, where `ande2"l"(a,b,a1)` : `b` : `[x:a]'prop'`. So $n = 2$.

```
@[a:'prop'][b:'prop'][a1:and(a,b)]

ande2"l":=...:b

a@[b:[x:a]'prop'][a1:and(a,b)]

ande2:=<ande1(...)>ande2"l"(a,b,a1):<ande1(...)>b
```

6. The extended applicability condition - Considerations

1. The example is not convincing: the extended applicability condition with $n > 1$ is useless in systems with three levels of terms, like Aut-QE.

- In fact we can remove it by replacing `b` with `[x:a]b` in four places; from the logical standpoint we are adding four missing $\forall$-introductions.

2. To us, extended applicability may help in two contexts: when $n = 0$ (Π-reduction), or when $n > 1$ in systems with many levels of terms.

- We do not know of any mathematics formalized in these contexts.

3. The literature about Λ_∞ and $\lambda\delta$-2 shows that the theory of a system supporting the extended applicability condition is not trivial at all.

- A mutual dependence arises between 1-step subject reduction and k-steps subject reduction, which involves other properties as well.

- This is solved by using a simultaneous induction on three axes.

7. The use of Π-reduction - Considerations

• When Π-reduction is in effect, we assign to the term $(N)(\Pi_x V)T$ the meaning of $[N/x]T$, and state that $(N)(\Pi_x V)T$ reduces to $[N/x]T$.

1. In a PTS, Π-reduction allows to remove substitution from the inferred type of $(N)M$, *i.e.*, if $M : (\Pi_x V)T$ then $(N)M : (N)(\Pi_x V)T$.

• Canonical type synthesis becomes syntax oriented and is decoupled from the reduction machinery, which is responsible for substitution.

• Environments with explicit substitutions may be needed in order to preserve the desired properties of the system (Kamareddine, 1996).

2. In the Grundlagen (Aut-QE), we need Π-reduction in cases such as: $(N)(\lambda_x V)M : (N)(\Pi_x V)T \to [N/x]T$ (typing plus reduction).

• This is not convincing: a PTS can do this without Π-reduction.

• Every Π-reduction needed to validate the Grundlagen is of this kind.

8. The weak correctness - Considerations

• Considering extended applicability for a PTS, weak correctness requires just the validity of $[N/x]T$ instead of the validity of $(\Pi_x V)T$.

1. The next example allows to compare these two forms of correctness: given $N : V : S$ and $M : T$, take the term $(V)(\lambda_a S)(N)(\lambda_x a)M$.

• This term is weakly correct (ΔΛ) but not strongly correct (PTS) because $[V/a]\underline{(N)(\lambda_x a)M}$ is valid, but $(\lambda_a S)\underline{(N)(\lambda_x a)M}$ is not.

2. A strongly correct term is weakly correct as well; conversely, a weakly correct term becomes strongly correct if reduced (de Bruijn, 1987).

• The test for weak correctness is easily implemented with de Bruijn's validation machines (ibid.), *i.e.*, state automata asserting correctness.

3. A straightforward translation of the Grundlagen is valid in CC (Brown, 2011; Guidi, 2015), so the Grundlagen is strongly correct.

Thank you

9. De Bruijn's validation machine for $\Delta\Lambda$ - Overview

• Testing correctness with a greedy approach, we may need to compute the same reduct more than once, as the next example clearly shows.

• Take the term $(N_2)(N_1)\underline{(N_0)(\lambda_{x_0}V_0)}(\lambda_{x_1}V_1)(\lambda_{x_2}V_2)M$. The redex (0) must be reduced when validating both applications (1) and (2).

• De Bruijn introduces a lazy algorithm that uses an argument stack. The original rules for $\Delta\Lambda$ follow. Warning: they test weak correctness.

1. $\langle \mathbf{R}, \epsilon, \star \rangle$ (final state)
2. $\langle \mathbf{R}, \mathbf{W}, \mathtt{typ}[x] \rangle$ implies $\langle \mathbf{R}, \mathbf{W}, x \rangle$
3. $\langle \mathbf{R}, \epsilon, N \rangle$ and $\langle \mathbf{R}, \mathbf{W}(N), \mathbf{B} \rangle$ implies $\langle \mathbf{R}, \mathbf{W}, (N)\mathbf{B} \rangle$
4. $\langle \mathbf{R}, \epsilon, V \rangle$ and $\langle \mathbf{R}(\flat_x V), \epsilon, \mathbf{B} \rangle$ implies $\langle \mathbf{R}, \epsilon, (\flat_x V)\mathbf{B} \rangle$
5. $\langle \mathbf{R}, \epsilon, V \rangle$ and $\langle \mathbf{R}(N)(\flat_x V), \mathbf{W}, \mathbf{B} \rangle$ implies $\langle \mathbf{R}, \mathbf{W}(N), (\flat_x V)\mathbf{B} \rangle$

   if $\mathbf{R} \vdash \mathtt{typ}[N] =_\beta V$. $\mathtt{typ}[N]$ is the syntax-oriented inf. type of N.

10. De Bruijn's validation machine for $\Delta\Lambda$ - Considerations

1. The state $\langle \mathbf{R}, \mathbf{W}, \mathbf{B} \rangle$ of the machine follows de Bruijn's original terminology: the "red part", the "white part", and the "blue part".

- If the machine $\langle \epsilon, \epsilon, \mathbf{B} \rangle$ reaches the final state, the term $\mathbf{B}$ is correct.
- De Bruijn adds the "yellow part" for a stack of trusted arguments.
- Be Bruijn does not use the terminology of machines. In particular, closures are not considered and α-conversion is assumed when needed.

2. These ideas might lead to design a lazy validation/type-checking algorithm for CIC, and to an implemented validation machine.

- A bidirectional validation algorithm for CIC (*i.e.*, `matita`) would employ a register holding the expected type of $\mathbf{B}$. Which color? :-)

3. We shall design and implement a validation machine for $\lambda\delta$-3 in `helena`, by which we expect the Grundlagen to validate faster.