The Formal System $\lambda\delta$

Ferruccio Guidi

University of Bologna, Italy

fguidi@cs.unibo.it

October 11, 2008

Leading conjecture

We can conceive a typed λ-calculus with the following features:

1. it should pursue the unification of terms, types and environments;
2. its meta-theory should include the commonly desired properties;
3. It should serve as a modular logical framework with enough flexibility to encode Mathematics in a realistic manner.

In this talk

- we give some motivations for desiring Feature 1;
- we discuss the main design issues of this calculus;
- we report on our attempt to realize this calculus by presenting a formal system that we call $\lambda\delta$ after the names of its binders.

Abbreviations in environments

- Mathematics is unimaginable without abbreviations;
- abbreviations are a crucial in optimal reduction strategies.

The unification of terms and types

Reverse Curry-Howard interpretation becomes possible as in:

- the realizability tradition (Kleene);
- the Automath experience (De Bruijn).

The unification of terms and environments

Environments and contexts become first-class objects:

- aggregates are possible without inductive types;
- the $\lambda\mu$ tradition (esp. Curien and Herbelin).

The set of the expressions

We use a reasonably small set of constructions,
which we plan to extend in the future.

The starting point is Λ_∞ (van Benthem Jutting)
[one sort, variables, binary applications, typed λ-abstractions].

We apply the following modifications to this basic platform:

- we add abbreviations, like let $x = v$ in t
  (benefit: we have non-recursive definitions);
- we add an infinite number of sorts
  (benefit: every legal term will have a type);
- we add explicit type annotations, like $(t : u)$
  (benefit: type checking easily reduces to type inference).

The set of the reduction schemes

For now we want deterministic and confluent computations.

We consider the following reduction schemes that work on terms, but that are designed to work on environments as well when possible.

- β-contraction: in call-by-name style as opposed to Λ_∞
  (benefit: this scheme works on environments as well as on terms);
- δ-expansion: unfolds an abbreviation without removing it;
- ζ-contraction: removes an unreferenced abbreviation
  (benefit: this scheme allows the call-by-value β-contraction);
- υ-swap: permutes an application-abbreviation pair
  (benefit: this scheme is used in optimal reduction strategies);
- τ-contraction: removes an explicit type annotation.

The type system

We confine the dynamic aspect of typing in the "conversion" rule (benefit: we separate construction and conversion during inference).

- Sorts: we have a sequence $h \mapsto *h$ of sorts and $*h$ is typed by $*g(h)$ where g can be chosen at will as long as $\forall h.\ h < g(h)$.
- Variables: we combine the usual "start" and "weakening" rules.
- Abstractions: we use the typing policy of Λ_∞ (λ-typing).
- Abbreviations: we adapt the λ-typing pattern to abbreviations (benefit: we have a uniform typing policy for both binders).
- Applications: we adapt the "compatible application" rule of the λ-Cube with Π-reduction (benefit: no reduction is involved).
- Type annotations: we use a "compatible" typing policy as well.

The meta-theoretical properties

We proved the following main meta-theoretical results:

- the reduction is confluent (Church-Rosser property);
- the reduction is safe (subject reduction property);
- the typed terms are strongly normalizing;
- the type inference problem is decidable.

Our calculus was born and developed as a digital specification, which is not the formal counterpart of some informal material.

The digital specification was checked by COQ and MATITA.
See the Web site: `http://helm.cs.unibo.it/lambda-delta/`

Some design features simplify the digital specification significantly.

Current achievements

- We achieved the full unification between terms and types.
- Our calculus has a desirable meta-theory in the usual sense.
- Our calculus can justify the structural fragment of the type theories in the Marin-Löf style (esp. CTT and mTT).

References

- F. Guidi: *The Formal System lambda-delta.*
  To appear in ACM ToCL. CoRR identifier: arXiv:cs/0611040.

Current limitations

- For now we do not have the higher-order abstraction, the Π of shapes $(\Box, *)$ and $(\Box, \Box)$, so the expressive power is that of λP.
- For now a variable occurrence is not an environment constructor: interpreting $\lambda x{:}w.x$ as an environment is not straight-forward.
- The ζ-reduction scheme does not work on environments.
- Following Λ_∞, for now we do not consider the η-reduction.
- We strengthen the applicability condition with respect to Λ_∞: the problem $(x_0 : \lambda y{:}\tau.y), (x_1 : x_0), (z : \tau) \vdash (x_1\ z) : ?$ has a solution in Λ_∞ but has no solutions in our calculus.
- The meta-theoretical properties are not proved on paper yet. A hard copy of the digital specification should take 600 pages.

Notational conventions

In the suggested notation for $\lambda\delta$ we use the following conventions:

- i, j, h, k denote meta-variables for natural numbers;
- R, S, T, U, V, W denote meta-variables for terms and types;
- C, D, E, F denote meta-variables for environments and contexts;
- the local variables are referenced by position (in De Bruijn style) (benefit: the equivalence by α-conversion is implicit);
- the terms and the environments are displayed in "item notation" (benefit: we improve the visual understanding of the redexes).

We use capital meta-variables as in the untyped λ-calculus tradition.

The use of position indexes and of the item notation comes from Λ_∞.

Syntactical items

- $*h$ (sort of index h, starts at 0);
- $\#i$ (local reference of index i, starts at 0);
- λW (abstractor of the type W);
- δV (abbreviator of the term V);
- (V) (applicator of the term V);
- $\langle U \rangle$ (annotator of the type U).

Terms and types

$*h \mid \#i \mid \lambda W.T \mid \delta V.T \mid (V).T \mid \langle U \rangle.T$ (with syntactic equality)

Environments and contexts

$*h \mid \lambda W.E \mid \delta V.E \mid (V).E \mid \langle U \rangle.E$ (with syntactic equality)

Native Type Assignment

$$\frac{}{E \vdash_g *h : *g(h)}\ \text{sort}$$

$$\frac{{}_0\!\downarrow^i E = C.\delta V \quad C \vdash_g V : W}{E \vdash_g \#i : {}_0\!\uparrow^{i+1} W}\ \text{def} \qquad \frac{{}_0\!\downarrow^i E = C.\lambda W \quad C \vdash_g W : R}{E \vdash_g \#i : {}_0\!\uparrow^{i+1} W}\ \text{decl}$$

$$\frac{E \vdash_g V : W \quad E.\delta V \vdash_g T : U}{E \vdash_g \delta V.T : \delta V.U}\ \text{abbr} \qquad \frac{E \vdash_g W : R \quad E.\lambda W \vdash_g T : U}{E \vdash_g \lambda W.T : \lambda W.U}\ \text{abst}$$

$$\frac{E \vdash_g V : W \quad E \vdash_g T : \lambda W.U}{E \vdash_g (V).T : (V).\lambda W.U}\ \text{appl} \qquad \frac{E \vdash_g T : U \quad E \vdash_g U : S}{E \vdash_g \langle U \rangle.T : \langle S \rangle.U}\ \text{cast}$$

$$\frac{E \vdash_g U_2 : S \quad E \vdash_g T : U_1 \quad E \vdash U_1 \Leftrightarrow^* U_2}{E \vdash_g T : U_2}\ \text{conv}$$