An Efficient Validation Procedure for the Formal System $\lambda\delta$

Ferruccio Guidi

University of Bologna, Italy

fguidi@cs.unibo.it

June 25, 2010

Overview

- The system $\lambda\delta$ "*brg*" is a typed λ-calculus inspired by the system Λ_∞.
- $\lambda\delta$ "*brg*" terms have sorts: $*l$, variables, typed abstractions: $\lambda W.T$, abbreviations: $\delta V.T$, applications: $(V).T$, and type annotations: $\langle U \rangle.T$.
- Conversion and typing occur in a context made of declarations and definitions. Both can be local (ref. by index) or global (ref. by name).
- Reductions include: call-by-name β-contraction, local and global δ-expansion, ζ-contraction, and type annotation removal (τ-contraction).
- The typing policy is "compatible": every construct that is not a variable, is typed by a construct of the same kind (implies λ-typing).
- We allow the PTS-style conversion rule and the pure application rule:

$$\frac{\mathcal{G}, E \vdash_h T : U \quad \mathcal{G}, E \vdash_h (V).U : W}{\mathcal{G}, E \vdash_h (V).T : (V).U} \text{pure} \quad \text{(N.G. de Bruijn, 1991)}$$

Overview (continued)

- We are also interested in this reduction, which is not part of $\lambda\delta$ "*brg*":

sort inclusion: $\lambda W.{*}l \to {*}l$ (I. Zandleven, 1973).

It gives $\lambda\delta$ the expressive power of λP, but it can not be applied freely.

- A global context is valid if all terms it contains are typable, and we are proposing an efficient algorithmic procedure to verify this property.

- $\lambda\delta$ resembles a PTS enough to address the problem of efficient type inference by means of a suitable extension of the Constructive Engine.

1. Our convertibility checker operates on two closures of possibly different length, rather than on two terms closed in a common context.

2. We use full reduction engines in place of normal closures throughout the type synthesizer and throughout the convertibility checker.

- We assert the applicability cond. without extracting the w.h.n.f. of the type of the function from the reduction machine that computed it.

The Reduction and Typing Machine

- The RTM is a KN machine that does not evaluate the stack contents, but that can compute the w.h.n.f. of the iterated type of a term.

- The RTM state $(a, \mathcal{G}, \mathcal{E}, \mathcal{S}, T)$ includes a level indicator, a global context, a local context of closures, a stack of closures, and the code.

- The local context contains "normal" entries as well as "special" entries λ^a (corresponding to the $V(a+1)$ entries of the KN machine).

1. In the *convertibility* mode, the RTM stops on sorts, references to the global context, references to local declarations and on abstractions.

2. In the *applicability* mode, the RTM stops on sorts and abstractions because the following transitions are enabled ("reference typing"):

$$(a, \mathcal{G}, E.\lambda^b(\mathcal{F}, W), \mathcal{S}, \#0) \ \rightarrow_{\text{local r.t.}} \ (a, \mathcal{G}, \mathcal{F}, \mathcal{S}, W)$$

$$(a, \mathcal{G}_1.\lambda_x W.\mathcal{G}_2, \mathcal{E}, \mathcal{S}, \$x) \ \rightarrow_{\text{global r.t.}} \ (a, \mathcal{G}_1.\lambda_x W.\mathcal{G}_2, \mathcal{E}, \mathcal{S}, W)$$

The convertibility test

- The test operates on two types closed in the respective contexts, given the invariant that they contain the same number of abstractions.

- The types are reduced in parallel by two RTMs running in the *convertibility* mode, and are compared each time a w.h.n.f. is reached.

- Two local references are compared by level (i.e., the a of the λ^a they refer to) so they do not need to be relocated before the comparison.

- A heuristic to avoid some useless global δ-expansions is implemented. Note that the test is not symmetric when sort inclusion is in effect.

- When sort inclusion is in effect, it must be tested as a last resort before asserting that the compared types are not convertible.

- S.i. is disabled when matching the RTMs stacks and the domains of the abstractions, otherwise some non-normalizing terms (Ω) are valid.

Type synthesis

- We improve the efficiency of the standard algorithm by testing the applicability cond. as shown by the following fragment of Caml code.

```
(* m: rtm, u: type of the function, w: type of the argument *)
let assert_applicability st m u w = match xwhd st m u with
   | _, Sort _                  -> error ...     (* sort case *)
   | mu, Bind (_, Abst u, _) ->                  (* abstraction case*)
     if are_convertible st mu u m w then () else error ...
   | _                          -> assert false (* impossible case *)
```

- `mu` and `u` go from `xwhd` to `are_convertible` as they are.

- Passing two RTMs to `are_convertible` is crucial here.

- `xwhd` computes the w.h.n.f. of the type of the function running the RTM in the *applicability* mode to take the *pure* type rule into account.

- `are_convertible` performs the convertibility test.

- `st` contains a user-set flag that activates sort inclusion on request.

Type synthesis (continued)

- The type U of a variable x is always inferred in the context where x is introduced, which may differ from the contexts in which x is invoked.

- Therefore, we need to relocate the de Bruijn indexes of U during type synthesis. It should be possible to avoid this time-consuming operation.

Testing the validation procedure

- We implemented our procedure as part of the HELM software.

- Enabling sort inclusion, we validated a two-steps naive mechanical translation of Jutting's "*Grundlagen der Analysis*" into $\lambda\delta$ "*brg*".

- In the first step we build an intermediate representation where the syntactic shorthand is removed, then we encode this into $\lambda\delta$ "*brg*".

- Unfortunately, the only competing validator for the "*Grundlagen*" is written in C rather than in Caml, so a comparison would not be fare.

Some statistical data

Size of the "*Grundlagen*"	
Language	*Int. complexity*
Aut − QE	319706
intermediate	754578
$\lambda\delta$ "*brg*"	998232

Performance of the validator		
Phase	*Run time fraction*	*Run time*
parsing	10%	0.7s
translation	25%	1.7s
validation	65%	4.4s

Relocated data	
terms	295202
int. complexity	1252256
a relocation occurs when the type of a local reference is computed	

Reductions			
β	1034626	τ	17166
local δ	494271	local r.t.	1
global δ	17166	global r.t.	0
ζ	0	s.i.	904

- The "*intrinsic complexity*" approximates the number of nodes.
  The validator was run on a 2×AMD Athlon MP 1800+, 1.53 GHz.
  The ζ-contractions, avoided by the validator, would be: 3694769.

Thank you

The abstract syntax of $\lambda\delta$ "*brg*"

Natural number: i,l,x (corresponding data-type: $\mathbb{N}$)

Term: $T,U,V,W ::= *l \mid \#i \mid \$x \mid \langle U \rangle.T \mid (V).T \mid \lambda W.T \mid \delta V.T$

Local environment: $E ::= * \mid E.\lambda W \mid E.\delta V$

Global environment: $\mathcal{G} ::= * \mid \mathcal{G}.\lambda_x W \mid \mathcal{G}.\delta_x V$

The reduction steps of $\lambda\delta$ "*brg*"

$\mathcal{G},\ E \vdash (V).\lambda W.T \to_\beta \delta V.T$	$\mathcal{G},\ E \vdash \langle U \rangle.T \to_\tau T$
$\mathcal{G},\ E_1.\delta V.E_2 \vdash \#i \to_\delta \uparrow^{i+1} V$ if $i = \lvert E_2 \rvert$	$\mathcal{G}_1.\delta_x V.\mathcal{G}_2,\ E \vdash \$x \to_\delta V$ if $x \notin \mathcal{G}_2$
$\mathcal{G},\ E \vdash \delta V.\uparrow^1 T \to_\zeta T$	$\mathcal{G},\ E \vdash (V_1).\delta V_2.T \to_\upsilon \delta V_2.(\uparrow^1 V_1).T$

$\uparrow^i$ is the "*relocation function*". $\lvert E_2 \rvert$ is the number of binders in E_2.

$x \notin \mathcal{G}_2$ means that there is no global binder named x in $\mathcal{G}_2$.

The fundamental judgements of $\lambda\delta$ "*brg*"

- $h : \mathbb{N} \to \mathbb{N}$ is any function satisfying $h(l) > l$ for each l.
- Conversion: $\mathcal{G}, E \vdash U_1 \leftrightarrow^* U_2$ (U_1 and U_2 are convertible).
- Type assignment: $\mathcal{G}, E \vdash_h T : U$ (T has type U).
- Correctness: $\mathrm{wf}_h(\mathcal{G})$ ($\mathcal{G}$ is well formed).

The type assignment rules of $\lambda\delta$ "*brg*"

$$\frac{\mathcal{G}_1, * \vdash_h V : W \quad x \notin \mathcal{G}_2}{\mathcal{G}_1.\delta_x V.\mathcal{G}_2, E \vdash_h \$x : W}\ \mathrm{g{-}def} \qquad \frac{\mathcal{G}_1.* \vdash_h W : V \quad x \notin \mathcal{G}_2}{\mathcal{G}_1.\lambda_x W.E_2, E \vdash_h \$x : W}\ \mathrm{g{-}decl}$$

$$\frac{\mathcal{G}, E_1 \vdash_h V : W \quad i = |E_2|}{\mathcal{G}, E_1.\delta V.E_2 \vdash_h \#i : \uparrow^{i+1} W}\ \mathrm{l{-}def} \qquad \frac{\mathcal{G}, E_1 \vdash_h W : V \quad i = |E_2|}{\mathcal{G}, E_1.\lambda W.E_2 \vdash_h \#i : \uparrow^{i+1} W}\ \mathrm{l{-}decl}$$

The type assignment rules of $\lambda\delta$ "*brg*" (continued)

$$\frac{}{\mathcal{G}, E \vdash_h *l : *h(l)}\ \text{sort} \qquad \frac{\mathcal{G}, E \vdash_h T : U \quad \mathcal{G}, E \vdash_h U : V}{\mathcal{G}, E \vdash_h \langle U \rangle.T : \langle V \rangle.U}\ \text{cast}$$

$$\frac{\mathcal{G}, E \vdash_h V : W \quad \mathcal{G}, E.\delta V \vdash_h T : U}{\mathcal{G}, E \vdash_h \delta V.T : \delta V.U}\ \text{abbr} \qquad \frac{\mathcal{G}, E \vdash_h W : V \quad \mathcal{G}, E.\lambda W \vdash_h T : U}{\mathcal{G}, E \vdash_h \lambda W.T : \lambda W.U}\ \text{abst}$$

$$\frac{\mathcal{G}, E \vdash_h V : W \quad \mathcal{G}, E \vdash_h T : \lambda W.U}{\mathcal{G}, E \vdash_h (V).T : (V).\lambda W.U}\ \text{appl} \qquad \frac{\mathcal{G}, E \vdash_h T : U \quad \mathcal{G}, E \vdash_h (V).U : W}{\mathcal{G}, E \vdash_h (V).T : (V).U}\ \text{pure}$$

$$\frac{\mathcal{G}, E \vdash_h U_2 : V \quad \mathcal{G}, E \vdash_h T : U_1 \quad \mathcal{G}, E \vdash U_1 \leftrightarrow^* U_2}{\mathcal{G}, E \vdash_h T : U_2}\ \text{conv}$$

The correctness rules of $\lambda\delta$ "*brg*"

$$\frac{}{\text{wf}_h(*)}\ \text{sort} \qquad \frac{\text{wf}_h(\mathcal{G}) \quad \mathcal{G}, * \vdash_h V : W}{\text{wf}_h(\mathcal{G}.\delta V)}\ \text{abbr} \qquad \frac{\text{wf}_h(\mathcal{G}) \quad \mathcal{G}, * \vdash_h W : V}{\text{wf}_h(\mathcal{G}.\lambda W)}\ \text{abst}$$

The Reduction and Typing Machine (supplement)

- The RTM state $(a, \mathcal{G}, \mathcal{E}, \mathcal{S}, T)$ has the following detailed structure:

$$a \in \mathbb{N}; \quad \mathcal{E} \ ::= \ * \mid \mathcal{E}.\lambda^a\mathcal{C} \mid \mathcal{E}.\delta\mathcal{C}; \quad \mathcal{S} \ ::= \ * \mid \mathcal{S}.\mathcal{C}; \quad \mathcal{C} \ ::= \ (\mathcal{E}, T)$$

- The RTM initial state is: $\mathcal{I}(\mathcal{G}, T) \equiv (0, \mathcal{G}, *, *, T)$.

- We provide for a read/push access to the RTM context because we want to use it as a reduction and type synthesis context as well.

- The RTM controllers force this reduction to cross a λ-abstraction:

$$(a, \mathcal{G}, \mathcal{E}, *, \lambda W.T) \ \rightarrow_{\text{push}} \ (a + 1, \mathcal{G}, \mathcal{E}.\lambda^a(\mathcal{E}, W), *, T)$$

- The RTM context $(\mathcal{E})$ accepts pushing only if the stack $(\mathcal{S})$ is empty.

- Formally, "reference typing" follows the pattern of δ-expansion, so the RTM does not need to perform any relocation when computing it.

- We implement "sort inclusion" and "reference typing" as reduction steps just for the type-synthesis algorithm. They break $\lambda\delta$'s theory.

How the RTM applies the “pure” type rule

- The term $(V).T$ is typable in $\mathcal{G}$ and E if (V) matches a λW_1 found in T, E or $\mathcal{G}$. Moreover W_1 and the type W_2 of V must be convertible.
- The item λW_1 must start the w.h.n.f. of the type U of T, or else, if the “*pure*” rule is in effect, the iterated types of U must be considered.
- This search eventually comes to an end since it involves just a finite number of iterated types of T, which are strongly normalizable.
- If $T \equiv X.\#i$ is typed (where X denotes a term segment) and if $\#i$ refers to a λ-abstraction of type W, then $U \equiv X.\uparrow^{i+1}W$ is a type for T.
- When the RTM is started on T and has scanned the segment X, so that $\#i$ is in the code register, then it must compute a w.h.n.f. of U.
- As the segment X was scanned already, we just apply “reference typing” to continue the computation with W in the code register.