Extracting programs from proofs

Helmut Schwichtenberg

Mathematisches Institut, LMU, München

JAIST, 7. March 2014
Overview

- Parsing balanced lists of parentheses
  - Informal proof
  - Discussion of the extracted term
  - Formalization, extraction and testing
- Ishihara’s trick
- Computing with infinite data
The Dyck language of balanced lists of $L$ and $R$

$E$: expressions formed as lists of left and right parentheses $L$, $R$. Dyck language of balanced parentheses is generated by either of

grammar $U$:

$$E ::= \text{Nil} \mid ELER$$

grammar $S$:

$$E ::= \text{Nil} \mid LER \mid EE$$

Restrict attention to $U$ (has unique generation trees).
- Parsing balanced lists of parentheses
  - Informal proof
  - Discussion of the extracted term
  - Formalization, extraction and testing
- Ishihara’s trick
- Computing with infinite data
Goal: recognize whether a list of left and right parentheses is balanced, and if so produce a generating tree (i.e., a parse tree).

- **Write-and-verify** method: write a parser as a shift-reduce syntax analyser, and verify that it is correct and complete.

- **Prove-and-extract** method: Prove the specification $A$ and extract its computational content in the form of a realizing term $t$. Since $t$ is in $T^+$, we can automatically prove (verify) $tr\ A$, by means of a formalization of the soundness theorem.
Formulate the grammar $U$ as an inductively defined predicate over lists $x, y, z$ of parentheses $L, R$ given by the clauses

- **InitU**: $U(\text{Nil})$
- **GenU**: $Ux \rightarrow Uy \rightarrow U(xLyR)$

Work with $RP(n, x)$ meaning $U(xR^n)$ and $LP(n, y)$ meaning $U(L^n y)$. For $RP$ we have an inductive definition

- $RP(0, \text{Nil})$
- $Uz \rightarrow RP(n, x) \rightarrow RP(n + 1, xzL)$

$LP$ can be defined via a boolean valued function

- $LP(0, \text{Nil}) = \text{tt}$
- $LP(n + 1, \text{Nil}) = \text{ff}$
- $LP(n, Lx) = LP(n + 1, x)$
- $LP(0, Rx) = \text{ff}$
- $LP(n + 1, Rx) = LP(n, x)$
Closure property of $U$

$$\forall_y \forall_{n,x,z} (\text{RP}(n, x) \rightarrow^c Uz \rightarrow^c \text{LP}(n, y) \rightarrow U(xzy)).$$

**Proof.**

Show by induction on $y$ that the claim holds for all $n$.

Base Nil. Use elimination for $\text{RP}(n, x)$.

Step. In case $L :: y$ use $\text{IH}y$ for $n + 1$.

In case $R :: y$ again use elimination for $\text{RP}(n, x)$.

The first $\text{RP}$ clause uses Efq, the second one $\text{IH}y$, GenU and equality arguments.
Have

$$\forall^c_y \forall^{nc}_{n,x,z} (\text{RP}(n, x) \rightarrow^c Uz \rightarrow^c \text{LP}(n, y) \rightarrow U(xzy)).$$

- In particular $$\forall^c_y (\text{LP}(0, y) \rightarrow Uy).$$
- Conversely $$\forall_y (Uy \rightarrow \text{LP}(0, y))$$ (by elimination for $$U$$).
- Hence the test $$\text{LP}(0, y)$$ is correct (all $$y$$ in $$U$$ satisfies it) and complete (it implies $$y$$ in $$U$$).
- Because of $$\text{LP}(0, y) \iff Uy$$ we have a decision procedure for $$U$$. With $$p$$ a boolean variable we can express this by a proof of

$$\forall^c_y \exists^d_p ((p \rightarrow Uy) \land^1 ((p \rightarrow \text{F}) \rightarrow Uy \rightarrow \text{F})).$$

The computational content of this proof is a parser for $$U$$. Given $$y$$ it returns a boolean saying whether or not $$y$$ is in $$U$$, and if so it also returns a generation tree (i.e., a parse tree) for $$Uy$$. 
[x] LP 0 x@

(Rec list par=>list bin=>bin=>bin)x
([as,a][case as ((Nil bin) -> a)
  (a0::as0 -> 0)])
([par,x0,f,as,a]
  [case par
   (L -> f(a::as)0)
   (R -> [case as ((Nil bin) -> 0)
    (a0::as0 -> f as0(a0 B a))])])
(Nil bin)
0
- Parsing balanced lists of parentheses
  - Informal proof
  - Discussion of the extracted term
  - Formalization, extraction and testing
- Ishihara’s trick
- Computing with infinite data
It amounts to applying a function $g$ to $x$, Nil and $O$, where

$$g(\text{Nil}, \mathcal{A}, a) = \begin{cases} 
  a & \text{if } \mathcal{A} = \text{Nil} \\
  O & \text{else}
\end{cases}$$

$$g(L :: x_0, \mathcal{A}, a) = g(x_0, a :: \mathcal{A}, O)$$

$$g(R :: x_0, \mathcal{A}, a) = \begin{cases} 
  O & \text{if } \mathcal{A} = \text{Nil} \\
  g(x_0, \mathcal{A}_0, a_0 B a) & \text{if } \mathcal{A} = a_0 :: \mathcal{A}_0
\end{cases}$$
\[
g(Nil, s, a) = \begin{cases} 
a & \text{if } s = \text{Nil} \\
O & \text{else}
\end{cases}
\]
\[
g(L :: x_0, s, a) = g(x_0, a :: s, O)
\]
\[
g(R :: x_0, s, a) = \begin{cases} 
O & \text{if } s = \text{Nil} \\
g(x_0, s_0, a_0 B a) & \text{if } s = a_0 :: s_0
\end{cases}
\]

In \( g(x, s, a) \)

- \( x \) is a list of parentheses \( L, R \) to be parsed.
- \( s \) is a stack of parse trees.
- \( a \) is the working memory of the parser which stores the parse tree being generated.

Initially \( g \) is called with \( x \), the empty stack \( \text{Nil} \) and the empty parse tree \( O \).
\[
\begin{align*}
g(\text{Nil}, \text{a} \cdot, a) &= \begin{cases} 
a & \text{if } \text{a} \cdot = \text{Nil} \\
O & \text{else} 
\end{cases} \\
g(L :: x_0, \text{a} \cdot, a) &= g(x_0, a :: \text{a} \cdot, O) \\
g(R :: x_0, \text{a} \cdot, a) &= \begin{cases} 
O & \text{if } \text{a} \cdot = \text{Nil} \\
g(x_0, \text{a} \cdot_0, a_0 B a) & \text{if } \text{a} \cdot = a_0 :: \text{a} \cdot_0
\end{cases}
\end{align*}
\]

- Read \( x \) from left to right.
- Suppose \( x = L :: x_0 \). Push the current parse tree \( a \) (corresponding to \( E_0 \) in \( E_0 LE_1 R \)) onto the stack. Then \( g \) starts generating a parse tree for the rest \( x_0 \) of \( x \), with \( O \) in its working memory.
- Suppose \( x = R :: x_0 \). If the stack is \( \text{Nil} \), return \( O \). If not, pop the top element \( a_0 \) from the stack. Then \( g \) starts generating a parse tree for the rest \( x_0 \) of \( x \), the tail \( \text{a} \cdot_0 \) of the stack, and as current parse tree \( a_0 B a \) in its working memory.
Parsing balanced lists of parentheses
  ▶ Informal proof
  ▶ Discussion of the extracted term
  ▶ Formalization, extraction and testing

▪ Ishihara’s trick

▪ Computing with infinite data
(load "~/minlog/init.scm")

(add-algs "bin"
  '("bin" "0")
  '("bin=>bin=>bin" "BinBranch"))

(add-infix-display-string "BinBranch" "B" 'pair-op)

(set! COMMENT-FLAG #f)
(libload "nat.scm")
(libload "list.scm")
(set! COMMENT-FLAG #t)

(add-algs "par" '("L" "par") '("R" "par"))
(add-totality "par")

(add-var-name "p" (py "boole"))
(add-var-name "x" "y" "z" (py "list par"))
(add-ids
  (list (list "U" (make-arity (py "list par")) "bin"))
    '("U(Nil par)" "InitU")
    '("allnc x,y(U x -> U y -> U(x++L: ++y++R:))" "GenU")
  )

(add-program-constant "LP" (py "nat=>list par=>boole"))

(add-computation-rules
  "LP 0(Nil par)"         "True"
  "LP(Succ n)(Nil par)"  "False"
  "LP n(L::x)"           "LP(Succ n)x"
  "LP 0(R::x)"           "False"
  "LP(Succ n)(R::x)"     "LP n x" )
;; RP (with a parameter predicate to be substituted by U)

(add-pvar-name "P" (make-arity (py "list par"))

(add-ids
  (list (list "RP" (make-arity (py "nat") (py "list par")))
       "list")
  '("RP 0(Nil par)" "InitRP")
  '("allnc n,x,z(P z -> RP n x -> RP(Succ n)(x++z++L:))" "GenRP")
)
;; ClosureU
(set-goal
"all y allnc n,x,z(
(RP (cterm (x^) U x^))n x -> U z -> LP n y ->
U(x++z++y))")

;; Soundness
(set-goal "allnc y(U y -> LP 0 y)")

;; Completeness
(set-goal "all y(LP 0 y -> U y)")

;; ParseLemma
(set-goal "all y ex p((p -> U y) &
   ((p -> F) -> U y -> F))")
(animate "ClosureU")
(animate "Completeness")

(add-var-name "a" (py "bin"))
(add-var-name "as" (py "list bin"))
(add-var-name "f" (py "list bin=>bin=>bin"))

(define eterm (proof-to-extracted-term
    (theorem-name-to-proof "ParseLemma")))
(define parser-term (rename-variables (nt eterm)))
(ppc parser-term)
(test-parser-term parser-term 6)

Testing on L::R::R::R::R::R::R: No
Testing on L::L::R::R::R::R::R: No
Testing on L::R::L::R::R::R::R: No
Testing on L::L::L::R::R::R::R: Parse tree: O B O B O B O
Testing on L::R::R::L::R::R::R: No
Testing on L::L::R::L::R::R::R: Parse tree: O B(O B O)B O
Testing on L::R::L::L::R::R::R: Parse tree: (O B O)B O B O
Testing on L::L::L::L::R::R::R: No
Testing on L::R::R::R::L::R::R: No
Testing on L::L::R::R::L::R::R: Parse tree: (O B O)B O B O
Testing on L::R::L::L::L::R::R: No
Testing on L::L::L::L::L::R::R: No
Testing on L::R::L::L::L::L::R: No
Testing on L::L::L::L::L::L::R: No
Testing on L::L::L::L::L::L::R: No
Testing on L::L::L::L::L::L::R: No
- Parsing balanced lists of parentheses
  - Informal proof
  - Discussion of the extracted term
  - Formalization, extraction and testing
- Ishihara’s trick
- Computing with infinite data
Theorem (Ishihara’s trick)

Let $f$ be a linear map from a Banach space $X$ into a normed space $Y$, and let $(u_n)$ be a sequence in $X$ converging to 0. Then for $0 < a < b$ either $a \leq \|fu_n\|$ for some $n$ or $\|fu_n\| \leq b$ for all $n$.

Proof. Let $M$ be a modulus of convergence of $(u_n)$ to 0; assume $M0 = 0$. Call $m$ a hit on $n$ if $M_n \leq m < M_{n+1}$ and $a \leq \|fu_m\|$. First goal: define a function $h: \mathbb{N} \to \mathbb{N}$ such that

- $h_n = 0$ if for all $n' \leq n$ there is no hit;
- $h_n = m + 2$ if at $n$ for the first time we have a hit, with $m$;
- $h_n = 1$ if there is an $n' < n$ with a hit.
We will need the bounded least number operator $\mu_{ng}$ defined recursively as follows ($g$ a variable of type $\mathbb{N} \rightarrow \mathbb{B}$).

$$
\mu_0g := 0,
$$

$$
\mu_Sng := \begin{cases} 
0 & \text{if } g0 \\
S\mu_n(g \circ S) & \text{otherwise.}
\end{cases}
$$

From $\mu_{ng}$ we define

$$
\mu^n_{n_0}g := \begin{cases} 
(\mu_{n-n_0}\lambda mg(m + n_0)) + n_0 & \text{if } n_0 \leq n \\
0 & \text{otherwise.}
\end{cases}
$$
To define $h$ we use a function $g$ of type $\mathbb{N} \to \mathbb{B}$ (to be defined from $\text{cApproxSplit}$) such that

$$
\begin{cases}
  a \leq \|fu_m\| & \text{if } gm \\
  \|fu_m\| \leq b & \text{otherwise}
\end{cases}
$$

Then we can define $h_n := H(g, M, n)$ where

$$
H(g, M, n) := \begin{cases}
  0 & \text{if } M_n \leq \mu_{M_n}g \text{ and } M_{n+1} \leq \mu_{M_{n+1}}g \\
  \mu_{M_n}^{M_{n+1}}g + 2 & \text{if } M_n \leq \mu_{M_n}g \text{ and } \mu_{M_n}^{M_{n+1}}g < M_{n+1} \\
  1 & \text{if } \mu_{M_n}g < M_n.
\end{cases}
$$
Next goal: define from $h$ a sequence $(v_n)$ in $X$ such that

- $v_n = 0$ if $h_n = 0$;
- $v_n = nu_m$ if $h_n = m + 2$;
- $v_n = v_{n-1}$ if $h_n = 1$.

Let $\xi$ be the type of elements of $X$, and $\mathcal{U}: \mathbb{N} \rightarrow \xi$ a variable.
Define $v_n := V_\xi(g, M, \mathcal{U}, n)$ where (writing $u_m$ for $\mathcal{U}(m)$)

$$V_\xi(g, M, \mathcal{U}, n) := \begin{cases} 
0 & \text{if } H(g, M, n) = 0 \\
nu_m & \text{if } H(g, M, n) = m + 2 \\
0 \text{ (arbitrary)} & \text{if } H(g, M, n) = 1 \text{ and } n = 0 \\
V_\xi(g, M, \mathcal{U}, n - 1) & \text{if } H(g, M, n) = 1 \text{ and } n > 0.
\end{cases}$$

One can show that $(v_n)$ has the properties listed above.
Next we show that \((v_n)\) is a Cauchy sequence with modulus 
\(N(k) := 2k + 1\), which satisfies 
\[
\frac{N(k) + 1}{2N(k)} \leq \frac{1}{2^k}.
\]

Since our goal is stable, we may employ arbitrary case distinctions (here: there is a hit / there is no hit).

By the assumed completeness of \(X\) we have a limit \(v\) of \((v_n)\). Pick \(n_0\) such that \(|fv| \leq n_0a\). Assume that there is a first hit at some \(n > n_0\), with value \(m\). Then \(v = v_n = nu_m\) and 
\[
na \leq n\|fu_m\| = n\|fu_m\| = \|f(nu_m)\| = \|fv\| \leq n_0a < na,
\]
a contradiction. Hence beyond this \(n_0\) we cannot have a first hit.

If \(\forall n \leq n_0 h_n = 0\) then there is no hit and we have \(|fu_n| \leq b\) for all \(n\). Otherwise there is a hit before \(n_0\), hence \(a \leq \|fu_n\|\) for some \(n\).
The computational content machine extracted from this proof is

\[\text{[f,us,M,a,a0,k]}\]
\[\text{[let g}\]
\[\text{([n]negb(cAC([n0]cApproxSplitBooleRat}
\text{a a0 lnorm(f(us n0))k)n))}
\[\text{[case (H g M}
\text{ (cRealPosRatBound}
\text{ lnorm(f((cXCompl xi)
\text{ ((V xi)g M us)
\text{ ([k0]abs(IntS(2*k0)max 0)))

\text{ a))}
\text{ (Zero -> False)
\text{ (Succ n -> True)])]}\]

Here \( H \) and \( V \) are the functionals defined above.
cAC is the computational content of the axiom of choice

\[(pp \ "AC")\]
all m ex boole (Pvar nat boole)\(^\sim\) m boole ->
ex g all m (Pvar nat boole)\(^\sim\) m(g m)

and hence the identity. cApproxSplitBooleRat and cRealPosRatBound are the computational content of lemmata

all a,b,x,k(Real x -> 1/2\(^\ast\)k\(<=\)b-a ->
ex boole((boole -> x\(<=\)b) andu ((boole -> F) -> a\(<=\)x)))

all x,a(Real x -> 0\(<\)a -> ex n x\(<=\)n*a)
Modifying the theorem by decorations

- In our formulation of Ishihara’s trick we have used the decorated disjunction $\lor^u$ (u for uniform) to express the final alternative.

- This means that the computational content of the lemma returns just a boolean, expressing which side of the disjunction holds, but not returning a witness for the existential quantifier in the left hand side, $\exists_n a \leq \|fu_n\|$.

- To change this use the “left” disjunction $\lor^1$ instead.

Then literally the same proof works.
Note that the required witness is obtained by an application of chFind, the computational content of a lemma HFind:

(\text{pp} \ "\text{HFind}\")
al\ g,M,n(M \ Zero=\text{Zero} \to (H \ g \ M \ n=\text{Zero} \to \text{F}) \to
\ \text{ex} \ n0,m(n0\leq n \& H \ g \ M \ n0=m+2))
- Parsing balanced lists of parentheses
  - Informal proof
  - Discussion of the extracted term
  - Formalization, extraction and testing
- Ishihara’s trick
- Computing with infinite data
Case study: uniformly continuous functions (U. Berger)

- Formalization of an abstract theory of (uniformly) continuous real functions $f: I \to I$ ($I := [-1, 1]$).

- Let $Cf$ express that $f$ is a continuous real function. Assume the abstract theory proves

  $$Cf \to \forall n \exists m \forall a \exists b (f[I_{a,m}] \subseteq I_{b,n}) \quad \text{with } I_{b,n} := [b - \frac{1}{2^n}, b + \frac{1}{2^n}]$$

  Then

  $$n \mapsto m \quad \text{modulus of (uniform) continuity ($\omega$)}$$
  $$n, a \mapsto b \quad \text{approximating rational function ($h$)}$$

Read\(_X\) and its witnesses

Inductively define a predicate \(\text{Read}_X\) of arity (\(\varphi\)) by the clauses

\[
\forall^n_{nc} \forall_d \left( f[I] \subseteq I \right) \rightarrow X(Out_d \circ f) \rightarrow \text{Read}_X f), \quad (\text{Read}_X)^+_0
\]

\[
\forall^n_{nc} (\text{Read}_X (f \circ \text{In}_{-1}) \rightarrow \text{Read}_X (f \circ \text{In}_0) \rightarrow \text{Read}_X (f \circ \text{In}_1) \rightarrow \text{Read}_X f). \quad (\text{Read}_X)^+_1
\]

where \(I_d = \left[ \frac{d-1}{2}, \frac{d+1}{2} \right] (d \in \{-1, 0, 1\})\) and

\[
(Out_d \circ f)(x) := 2f(x) - d, \quad (f \circ \text{In}_d)(x) := f\left(\frac{x + d}{2}\right).
\]

Witnesses for \(\text{Read}_X f\): total ideals in

\[
\mathcal{R}_\alpha := \mu_\xi (\text{Put}^{\mathcal{SD}}_{\alpha \rightarrow \xi}, \text{Get}^{\xi \rightarrow \xi \rightarrow \xi \rightarrow \xi})
\]

where \(\mathcal{SD} := \{-1, 0, 1\}\).
Nested inductive definition of a predicate $\text{Write}$ of arity ($\varphi$):

$\text{Write}(\text{Id}), \quad \forall_{f}^{\text{nc}} (\text{Read}_{\text{Write}} f \rightarrow \text{Write} f) \quad (\text{Id identity function}).$

Witnesses for $\text{Write} f$: total ideals in

$$\mathcal{W} := \mu_{\xi}(\text{Stop}^{\xi}, \text{Cont}^{R_{\xi} \rightarrow \xi}).$$

Define $\text{coWrite}$, a companion predicate of $\text{Write}$, by

$$\forall_{f}^{\text{nc}} (\text{coWrite} f \rightarrow \text{Eq}(f, \text{Id}) \lor \text{Read}_{\text{coWrite}} f). \quad (\text{coWrite})^{-}$$

Witnesses for $\text{coWrite} f$: $\mathcal{W}$-cototal $R_{\mathcal{W}}$-total ideals $t$. 
$W$-cototal $R_W$-total ideals

are possibly non well-founded trees $t$:

- Get-Put-part: well-founded,
- Stop-Cont-part: not necessarily well-founded.
\textbf{W-cototal} \(R_W\)-total ideals as stream transformers

View them as \textit{read-write machines}.

- Start at the root of the tree.
- At node \(\text{Put}_d t\), output the digit \(d\), carry on with the tree \(t\).
- At node \(\text{Get} \ t_{-1} \ t_0 \ t_1\), read a digit \(d\) from the input stream and continue with the tree \(t_d\).
- At node \(\text{Stop}\), return the rest of the input unprocessed as output.
- At node \(\text{Cont} \ t\), continue with the tree \(t\).

Output might be infinite, but \(R_W\)-totality ensures that the machine can only read finitely many input digits before producing another output digit.

The machine represents a continuous function.
**Cf** implies $\text{coWrite}\, f$: informal proof

The greatest-fixed-point axiom $(\text{coWrite})^+$ (coinduction) is

$$\forall_{nc}^{\text{nc}}(Q\, f \to \forall_{nc}^{\text{nc}}(Q\, f \to \text{Eq}(f, \text{Id}) \lor \text{Read}_{\text{coWrite}} \lor Q\, f) \to \text{coWrite}\, f).$$

**Theorem** [Type-1 u.c.f. into type-0 u.c.f.]. $\forall_{nc}^{\text{nc}}(C\, f \to \text{coWrite}\, f)$.

**Proof.** Assume $C\, f$. Use $(\text{coWrite})^+$ with competitor $C$. Suffices $\forall_{nc}^{\text{nc}}(C\, f \to \text{Eq}(f, \text{Id}) \lor \text{Read}_{\text{coWrite}} \lor C\, f)$. Assume $C\, f$, in particular $B_{m,2}f := \forall a \exists b(f[I_a,m] \subseteq I_b,2)$ for some $m$. Get rhs by Lemma 1.

**Lemma 1.** $\forall m \forall_{nc}^{\text{nc}}(B_{m,2}f \to C\, f \to \text{Read}_{\text{coWrite}} \lor C\, f)$.

**Proof.** Induction on $m$, using Lemma 2 in the base case.

**Lemma 2** [FindSD]. $\forall_{nc}^{\text{nc}}(B_{0,2}f \to \exists_d(f[I] \subseteq l_d))$.

**Proof.** Assume $B_{0,2}f$. Then $f[I_0,0] \subseteq I_{b,2}$ for some $b$, by definition of $B_{n,m}$. Have $b \leq -\frac{1}{4}$, $-\frac{1}{4} \leq b \leq \frac{1}{4}$ or $\frac{1}{4} \leq b$. Can determine either of $I_{b,2} \subseteq I_{-1}, I_{b,2} \subseteq I_0$ or $I_{b,2} \subseteq I_1$, hence $\exists_d(f[I] \subseteq l_d)$. 
(oh](CoRec (nat=>nat@@(rat=>rat))=>algwrite)oh
((oh0]Inr((Rec nat=>..[type]..)
  left(oh0(Succ(Succ Zero)))
  ([g,oh1] [let sd (cFindSd(g 0))
   (Put sd
    (InR([n]left(oh1(Succ n))@
      ([a]2*right(oh1(Succ n))a-SDToInt sd))))])
([n,st,g,oh1]
  Get
  (st([a]g((a+IntN 1)/2))
   ([n0]left(oh1 n0)@
    ([a]right(oh1 n0)((a+IntN 1)/2)))
  (st([a]g(a/2))([n0]left(oh1 n0)@
     ([a]right(oh1 n0)(a/2)))
  (st([a]g((a+1)/2))([n0]left(oh1 n0)@
     ([a]right(oh1 n0)((a+1)/2))))
right(oh0(Succ(Succ Zero)))
oh0))
Corecursion

The corecursion operator $^{\text{co}}R^\tau_W$ has type

$$\tau \rightarrow (\tau \rightarrow U + R_{W+\tau}) \rightarrow W.$$  

Conversion rule

$^{\text{co}}R^\tau_W NM \mapsto [\text{case } (MN)^{U+R(W+\tau)} \text{ of} \n\text{inl } _x \mapsto \text{Stop} | \n\text{inr } x \mapsto \text{Cont}(\text{M}^{W}_{R(W+\tau)})(\lambda p [\text{case } p^{W+\tau} \text{ of} \n\text{inl } y^W \mapsto y | \n\text{inr } z^\tau \mapsto {^{\text{co}}R^\tau_W zM}]) \n\text{w}^R(W+\tau)]$

with $M$ the map-operator.

- Here $\tau$ is $N \rightarrow N \times (Q \rightarrow Q)$, for pairs of $\omega: N \rightarrow N$ and $h: N \rightarrow Q \rightarrow Q$ (variable name oh).

- No termination; translate into Haskell for evaluation.
Conclusion

TCF (theory of computable functionals) as a possible foundation for exact real arithmetic.

- Simply typed theory, with “lazy” free algebras as base types ($\Rightarrow$ constructors are injective and have disjoint ranges).
- Variables range over partial continuous functionals.
- Constants denote computable functionals ($:= r.e.$ ideals).
- Minimal logic ($\rightarrow, \forall$), plus inductive & coinductive definitions.
- Computational content in abstract theories.
- Decorations ($\rightarrow^c, \forall^c$ and $\rightarrow^{nc}, \forall^{nc}$) to (i) allow abstract theory and (ii) remove unused data.
References

- U. Berger, From coinductive proofs to exact real arithmetic. CSL 2009.
- K. Miyamoto and H.S., Program extraction in exact real arithmetic. To appear, MSCS.
- K. Miyamoto, F. Nordvall Forsberg and H.S., Program extraction from nested definitions. ITP 2013.