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

$$\text{grammar U :} \qquad E ::= \mathrm{Nil} \mid ELER$$
$$\text{grammar S :} \qquad E ::= \mathrm{Nil} \mid LER \mid EE$$

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

Parsing

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) $t \mathbf{r} A$, 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

$$\mathrm{InitU}\colon U(\mathrm{Nil})$$
$$\mathrm{GenU}\colon Ux \to Uy \to U(xLyR)$$

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

$$\mathrm{RP}(0, \mathrm{Nil})$$
$$Uz \to \mathrm{RP}(n, x) \to \mathrm{RP}(n+1, xzL)$$

LP can be defined via a boolean valued function

$$\begin{aligned} \mathrm{LP}(0, \mathrm{Nil}) &= \mathtt{tt} \\ \mathrm{LP}(n+1, \mathrm{Nil}) &= \mathtt{ff} \\ \mathrm{LP}(n, Lx) &= \mathrm{LP}(n+1, x) \\ \mathrm{LP}(0, Rx) &= \mathtt{ff} \\ \mathrm{LP}(n+1, Rx) &= \mathrm{LP}(n, x) \end{aligned}$$

Closure property of U

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

Proof.

Show by induction on y that the claim holds for all n.
Base Nil. Use elimination for $\mathrm{RP}(n,x)$.
Step. In case $L :: y$ use IHy for $n+1$.
In case $R :: y$ again use elimination for $\mathrm{RP}(n,x)$.
The first RP clause uses Efq, the second one IHy, GenU and equality arguments. □

Have

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

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

$$\forall^{\mathrm{c}}_{y}\exists^{\mathrm{d}}_{p}((p \to Uy) \wedge^{\mathrm{l}} ((p \to \mathbf{F}) \to Uy \to \mathbf{F})).$$

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.

Extracted term

```
[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

```
[x] LP 0 x@
 (Rec list par=>list bin=>bin=>bin)x
 ([as,a][case as ((Nil bin) -> a)
                 (a0::as0 -> O)])
 ([par,x0,f,as,a]
   [case par
     (L -> f(a::as)O)
     (R -> [case as ((Nil bin) -> O)
                    (a0::as0 -> f as0(a0 B a))])])
 (Nil bin)
 O
```

It amounts to applying a function g to x, Nil and O, where

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

$$g(L :: x_0, as, a) = g(x_0, a :: as, O)$$

$$g(R :: x_0, as, a) = \begin{cases} O & \text{if } as = \mathrm{Nil} \\ g(x_0, as_0, a_0\ B\ a) & \text{if } as = a_0 :: as_0 \end{cases}$$

$$
\begin{aligned}
g(\mathrm{Nil}, as, a) \quad &= \begin{cases} a & \text{if } as = \mathrm{Nil} \\ O & \text{else} \end{cases} \\
g(L :: x_0, as, a) &= g(x_0, a :: as, O) \\
g(R :: x_0, as, a) &= \begin{cases} O & \text{if } as = \mathrm{Nil} \\ g(x_0, as_0, a_0\ B\ a) & \text{if } as = a_0 :: as_0 \end{cases}
\end{aligned}
$$

In $g(x, as, a)$

- x is a list of parentheses L, R to be parsed.
- as 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 Nil and the empty parse tree O.

$$
\begin{aligned}
g(\mathrm{Nil}, as, a) \quad &= \begin{cases} a & \text{if } as = \mathrm{Nil} \\ O & \text{else} \end{cases} \\
g(L :: x_0, as, a) &= g(x_0, a :: as, O) \\
g(R :: x_0, as, a) &= \begin{cases} O & \text{if } as = \mathrm{Nil} \\ g(x_0, as_0, a_0 \; B \; a) & \text{if } as = a_0 :: as_0 \end{cases}
\end{aligned}
$$

- Read x from left to right.
- Suppose $x = L :: x_0$. Push the current parse tree a (corresponding to E_0 in $E_0 L E_1 R$) 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 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 as_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" "O")
  '("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: No
Testing on L::L::R::R::R::R: No
Testing on L::R::L::R::R::R: No
Testing on L::L::L::R::R::R: Parse tree: O B O B O B O
Testing on L::R::R::L::R::R: No
Testing on L::L::R::L::R::R: Parse tree: O B(O B O)B O
Testing on L::R::L::L::R::R: Parse tree: (O B O)B O B O
Testing on L::L::L::L::R::R: No
Testing on L::R::R::R::L::R: No
Testing on L::L::R::R::L::R: Parse tree: (O B O B O)B O
Testing on L::R::L::R::L::R: Parse tree: ((O B O)B O)B O
Testing on L::L::L::R::L::R: No
Testing on L::R::R::L::L::R: No
Testing on L::L::R::L::L::R: No
Testing on L::R::L::L::L::R: No
Testing on 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 $a \le \|fu_n\|$ for some n or $\|fu_n\| \le b$ 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 $M_n \le m < M_{n+1}$ and $a \le \|fu_m\|$. First goal: define a function $h \colon \mathbb{N} \to \mathbb{N}$ such that

- $h_n = 0$ if for all $n' \le n$ 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 $\mu_n g$ defined recursively as follows (g a variable of type $\mathbb{N} \to \mathbb{B}$).

$$\mu_0 g := 0,$$
$$\mu_{\mathrm{S}n} g := \begin{cases} 0 & \text{if } g0 \\ \mathrm{S}\mu_n(g \circ \mathrm{S}) & \text{otherwise.} \end{cases}$$

From $\mu_n g$ we define

$$\mu_{n_0}^{n} g := \begin{cases} (\mu_{n-n_0} \lambda_m g(m + n_0)) + n_0 & \text{if } n_0 \leq n \\ 0 & \text{otherwise.} \end{cases}$$

To define h we use a function g of type $\mathbb{N} \to \mathbb{B}$ (to be defined from `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

$$H(g, M, n) := \begin{cases} 0 & \text{if } M_n \leq \mu_{M_n} g \text{ and } M_{n+1} \leq \mu_{M_n}^{M_{n+1}} g \\ \mu_{M_n}^{M_{n+1}} g + 2 & \text{if } M_n \leq \mu_{M_n} g \text{ and } \mu_{M_n}^{M_{n+1}} g < M_{n+1} \\ 1 & \text{if } \mu_{M_n} g < M_n. \end{cases}$$

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

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

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

$$\frac{N(k)+1}{2^{N(k)}} \leq \frac{1}{2^k}.$$

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 $\|fv\| \leq n_0 a$. Assume that there is a first hit at some $n > n_0$, with value m. Then $v = v_n = nu_m$ and

$$na \leq n\|fu_m\| = \|n(fu_m)\| = \|f(nu_m)\| = \|fv\| \leq n_0 a < na,$$

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

```
[f,us,M,a,a0,k]
 [let g
   ([n]negb(cAC([n0]cApproxSplitBooleRat
                    a a0 lnorm(f(us n0))k)n))
   [case (H g M
          (cRealPosRatBound
           lnorm(f((cXCompl xi)
                   ((V xi)g M us)
                   ([k0]abs(IntS(2*k0)max 0))))
           a))
    (Zero -> False)
    (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)^ m boole ->
ex g all m (Pvar nat boole)^ 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**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 $\vee^{\mathrm{u}}$ (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 $\vee^{\mathrm{l}}$ instead.

Then literally the same proof works.

```
[f,us,M,a,a0,k]
 [let g
   ([n]negb(cAC([n0]cApproxSplitBooleRat
                    a a0 lnorm(f(us n0))k)n))
   [let n
    (cRealPosRatBound
    lnorm(f((cXCompl xi)
            ((V xi)g M us)
            ([k0]abs(IntS(2*k0)max 0))))
    a)
    [case (H g M n)
     (Zero -> (DummyR nat))
     (Succ n0 -> Inl right(cHFind g M n))]]]
```

Note that the required witness is obtained by an application of `cHFind`, the computational content of a lemma `HFind`:

```
(pp "HFind")
all g,M,n(M Zero=Zero -> (H g M n=Zero -> F) ->
 ex n0,m(n0<=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 $f : I \to I \quad (I := [-1, 1])$.
- Let $\mathrm{C}f$ express that f is a continuous real function. Assume the abstract theory proves

$$\mathrm{C}f \to \forall_n \exists_m \underbrace{\forall_a \exists_b (f[I_{a,m}] \subseteq I_{b,n})}_{\mathrm{B}_{m,n} f} \quad \text{with } I_{b,n} := [b - \tfrac{1}{2^n}, b + \tfrac{1}{2^n}]$$

Then

$n \mapsto m$	modulus of (uniform) continuity (ω)
$n, a \mapsto b$	approximating rational function (h)

Read_X and its witnesses

Inductively define a predicate Read_X of arity (φ) by the clauses

$$\forall_f^{\mathrm{nc}} \forall_d (f[I] \subseteq I_d \to X(\mathrm{Out}_d \circ f) \to \mathrm{Read}_X f), \qquad (\mathrm{Read}_X)_0^+$$

$$\forall_f^{\mathrm{nc}} (\mathrm{Read}_X(f \circ \mathrm{In}_{-1}) \to \mathrm{Read}_X(f \circ \mathrm{In}_0) \to \mathrm{Read}_X(f \circ \mathrm{In}_1) \to \mathrm{Read}_X f). \qquad (\mathrm{Read}_X)_1^+$$

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

$$(\mathrm{Out}_d \circ f)(x) := 2f(x) - d, \qquad (f \circ \mathrm{In}_d)(x) := f(\frac{x+d}{2}).$$

Witnesses for $\mathrm{Read}_X f$: total ideals in

$$\mathbf{R}_\alpha := \mu_\xi(\mathsf{Put}^{\mathbf{SD} \to \alpha \to \xi}, \mathsf{Get}^{\xi \to \xi \to \xi \to \xi})$$

where $\mathbf{SD} := \{-1, 0, 1\}$.

Write, ${}^{\mathrm{co}}\mathrm{Write}$ and its witnesses

Nested inductive definition of a predicate Write of arity (φ):

$\mathrm{Write}(\mathrm{Id})$, $\quad \forall^{\mathrm{nc}}_{f}(\mathrm{Read}_{\mathrm{Write}} f \to \mathrm{Write}\, f)$ $\qquad$ (Id identity function).

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

$$\mathbf{W} := \mu_{\xi}(\mathsf{Stop}^{\xi}, \mathsf{Cont}^{\mathbf{R}_{\xi} \to \xi}).$$

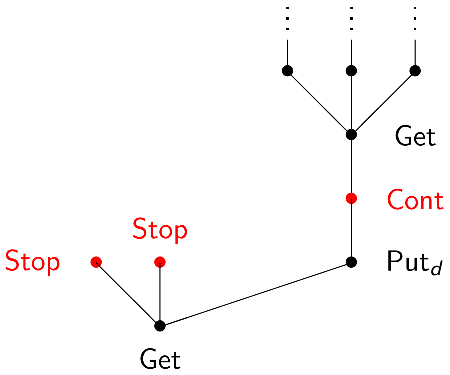
Define ${}^{\mathrm{co}}\mathrm{Write}$, a companion predicate of Write, by

$$\forall^{\mathrm{nc}}_{f}({}^{\mathrm{co}}\mathrm{Write}\, f \to \mathrm{Eq}(f, \mathrm{Id}) \lor \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}} f). \qquad ({}^{\mathrm{co}}\mathrm{Write})^{-}$$

Witnesses for ${}^{\mathrm{co}}\mathrm{Write}\, f$: $\mathbf{W}$-cototal $\mathbf{R}_{\mathbf{W}}$-total ideals t.

$\mathbf{W}$-cototal $\mathbf{R_W}$-total ideals

are possibly non well-founded trees t:



- Get-Put-part: well-founded,
- Stop-Cont-part: not necessarily well-founded.

$\mathbf{W}$-cototal $\mathbf{R_W}$-total ideals as stream transformers

View them as read-write machines.

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

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

The machine represents a continuous function.

$\mathrm{C}f$ implies ${}^{\mathrm{co}}\mathrm{Write}\, f$: informal proof

The greatest-fixed-point axiom $({}^{\mathrm{co}}\mathrm{Write})^+$ (coinduction) is

$$\forall^{\mathrm{nc}}_f(Q\,f \to \forall^{\mathrm{nc}}_f(Q\,f \to \mathrm{Eq}(f,\mathrm{Id}) \vee \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee Q} f) \to {}^{\mathrm{co}}\mathrm{Write}\, f).$$

Theorem [Type-1 u.c.f. into type-0 u.c.f.]. $\forall^{\mathrm{nc}}_f(\mathrm{C}f \to {}^{\mathrm{co}}\mathrm{Write}\, f)$.

Proof. Assume $\mathrm{C}f$. Use $({}^{\mathrm{co}}\mathrm{Write})^+$ with competitor C. Suffices $\forall^{\mathrm{nc}}_f(\mathrm{C}f \to \mathrm{Eq}(f,\mathrm{Id}) \vee \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee \mathrm{C}} f)$. Assume $\mathrm{C}f$, in particular $\mathrm{B}_{m,2}f := \forall_a \exists_b(f[I_{a,m}] \subseteq I_{b,2})$ for some m. Get rhs by Lemma 1.

Lemma 1. $\forall_m \forall^{\mathrm{nc}}_f(\mathrm{B}_{m,2}f \to \mathrm{C}f \to \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee \mathrm{C}} f)$.

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

Lemma 2 [FindSD]. $\forall^{\mathrm{nc}}_f(\mathrm{B}_{0,2}f \to \exists_d(f[I] \subseteq I_d))$.

Proof. Assume $\mathrm{B}_{0,2}f$. Then $f[I_{0,0}] \subseteq I_{b,2}$ for some b, by definition of $\mathrm{B}_{n,m}$. Have $b \le -\frac{1}{4}$, $-\frac{1}{4} \le b \le \frac{1}{4}$ or $\frac{1}{4} \le b$. Can determine either of $I_{b,2} \subseteq I_{-1}$, $I_{b,2} \subseteq I_0$ or $I_{b,2} \subseteq I_1$, hence $\exists_d(f[I] \subseteq I_d)$.

```
[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 ${}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{W}}$ has type

$$\tau \to (\tau \to \mathbf{U} + \mathbf{R}_{\mathbf{W}+\tau}) \to \mathbf{W}.$$

Conversion rule

$$\begin{aligned}
{}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{W}} NM \mapsto [&\textbf{case } (MN)^{\mathbf{U}+\mathbf{R}(\mathbf{W}+\tau)} \textbf{ of} \\
&\mathrm{inl}\ _ \mapsto \mathrm{Stop} \mid \\
&\mathrm{inr}\ x \mapsto \mathrm{Cont}(\mathcal{M}^{\mathbf{W}}_{\mathbf{R}(\mathbf{W}+\tau)}(\lambda_p[\textbf{case } p^{\mathbf{W}+\tau} \textbf{ of} \\
&\qquad\qquad \mathrm{inl}\ y^{\mathbf{W}} \mapsto y \mid \\
&\qquad\qquad \mathrm{inr}\ z^{\tau} \mapsto {}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{W}} zM]) \\
&\qquad x^{\mathbf{R}(\mathbf{W}+\tau)}]
\end{aligned}$$

with $\mathcal{M}$ the map-operator.

- Here τ is $\mathbf{N} \to \mathbf{N} \times (\mathbf{Q} \to \mathbf{Q})$, for pairs of $\omega\colon \mathbf{N} \to \mathbf{N}$ and $h\colon \mathbf{N} \to \mathbf{Q} \to \mathbf{Q}$ (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 ($\rightarrow^{\mathrm{c}}, \forall^{\mathrm{c}}$ and $\rightarrow^{\mathrm{nc}}, \forall^{\mathrm{nc}}$) 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.
- H.S. and S.S. Wainer, Proofs and Computations. Perspectives in Logic, ASL & Cambridge UP, 2012.