Program Extraction from Nested Definitions

Kenji Miyamoto[1,*], Fredrik Nordvall Forsberg[2,*,]**
and Helmut Schwichtenberg[1]

[1] Ludwig Maximilian University, Munich
[2] Swansea University, Wales

* Supported by the Marie Curie Initial Training Network in Mathematical Logic – MALOA – From MAthematical LOgic to Applications, PITN-GA-2009-238381.
** Supported by EPSRC grant EP/G033374/1, Theory and applications of induction-recursion.

26.07.2013
ITP 2013, Rennes

Outline

- Proof assistant Minlog and the theory TCF behind it to study computational meaning of proofs.
- Case study in exact real arithmetic.

Note on `listrev.scm`

We inductively define predicate A of arity $(\mathbf{L}_\mathbf{N}, \mathbf{L}_\mathbf{N}, \mathbf{L}_\mathbf{N})$. $A(u, v, w)$ means that the append of u and v is w.

$$\forall_v A([], v, v), \tag{A_0^+}$$

$$\forall_{u,v,w,x}(A(u, v, w) \to A(x::u, v, x::w)). \tag{A_1^+}$$

The above formulas are adopted as the *introduction axioms* of A.
We inductively define R of arity $(\mathbf{L}_\mathbf{N}, \mathbf{L}_\mathbf{N})$ as follows.

$$R([], []), \tag{R_0^+}$$

$$\forall_{u,v,w,x}(R(u, v) \to A(v, x::[], w) \to R(x::u, w)). \tag{R_1^+}$$

Note on `listrev.scm`

From the proof of the proposition $\forall_v \exists_w R(v, w)$ we extracted a term

$$\lambda_u(\mathcal{R}_{\mathbf{L_N}}^{\mathbf{L_N}}\, u\, []\, \lambda_{x,v,w}(\mathcal{R}_{\mathbf{L_N}}^{\mathbf{L_N}}\, w\, (x{::}[])\, \lambda_{y,\_}(y{::})))$$

of type $\mathbf{L_N} \to \mathbf{L_N}$.
We can export the term to Haskell.

```
module Main where

import Data.List

----- Algebras -----------------

type Nat = Integer

----- Recursion operators -------

listRec :: [alpha] -> alpha1 ->
                      (alpha -> ([alpha] -> (alpha1 -> alpha1))) ->
                      alpha1
listRec [] a f = a
listRec (b : z) a f = ((f b) z) (listRec z a f)
```

Note on listrev.scm

```
----- Program constants ---------

cLA :: [Nat] -> [Nat] -> [Nat]
cLA = \ v0 -> (\ v1 -> (listRec v1 v0 (\ x2 -> (\ v3 -> (:) x2))))

cLR :: [Nat] -> [Nat]
cLR = \ v0 -> (listRec v0 [] (\ x1 -> (\ v2 -> (cLA (x1 : [])))))

---------------------------------

rev :: [Nat] -> [Nat]
rev = cLR

apd :: [Nat] -> [Nat] -> [Nat]
apd = cLA

---------------------------------

main :: IO ()
main = putStrLn ""
```

Constants and axioms

The recursion operator $\mathcal{R}^{\rho}_{\mathbf{L}_\alpha}$ came from induction on lists.

$$\begin{aligned}
&\mathcal{R}^{\rho}_{\mathbf{L}_\alpha} : \mathbf{L}_\alpha \to \rho \to (\alpha \to \mathbf{L}_\alpha \to \rho \to \rho) \to \rho,\\
&\mathcal{R}^{\rho}_{\mathbf{L}_\alpha}\ []\ M_0\ M_1 = M_0,\\
&\mathcal{R}^{\rho}_{\mathbf{L}_\alpha}\ (x{::}u)\ M_0\ M_1 = M_1\ x\ u\ (\mathcal{R}^{\rho}_{\mathbf{L}_\alpha}\ u\ M_0\ M_1).
\end{aligned}$$

We relate $\mathcal{R}^{\rho}_{\mathbf{L}_\alpha}$ with the induction on list, which come from the *totality predicate* $T_{\mathbf{L}}$.

$$T_{\mathbf{L}}[], \qquad \forall^{\mathrm{nc}}_{x,u}(Q(x) \to T_{\mathbf{L}}(u) \to T_{\mathbf{L}}(x{::}u)), \qquad (T_{\mathbf{L}})^{+}_{0}, (T_{\mathbf{L}})^{+}_{1}$$

$$\forall^{\mathrm{nc}}_{u}(T_{\mathbf{L}} u \to P[] \to \forall^{\mathrm{nc}}_{x,u}(Q(x) \to T_{\mathbf{L}} u \to Pu \to P(x{::}u)) \to Pu). \qquad (T_{\mathbf{L}})^{-}$$

where Q is a parameter predicate of arity (α). We refer to $(T_{\mathbf{L}})^{-}$ by *elimination axiom* or induction.
We formally relate a term and a formula via realizability **r**.
For example, we expect:

- "Constructor" **r** "introduction axiom",
- "Recursion operator" **r** "elimination axiom",

Let A be a formula with proof M. We can compute:

- the type $\tau(A)$ of *potential realizers* of A.
- a *realizer* (extracted term) $\mathrm{et}(M)^{\tau(A)}$ of A (program extraction).

Realizability

Realizability is a way to think about a computational solution of a problem expressed by a formula.
We work in first-order minimal logic with implication and universal quantifiers. The realizability relation is:

$$t \mathbf{\ r\ } A \to B := \forall_x (x \mathbf{\ r\ } A \to t(x) \mathbf{\ r\ } B), \qquad t \mathbf{\ r\ } \forall_x A := \forall_x (t(x) \mathbf{\ r\ } A).$$

We consider non-computational variants of $\to$ and $\forall$.

$$t \mathbf{\ r\ } A \to^{\mathrm{nc}} B := \forall_x (x \mathbf{\ r\ } A \to t \mathbf{\ r\ } B), \qquad t \mathbf{\ r\ } \forall_x^{\mathrm{nc}} A := \forall_x (t \mathbf{\ r\ } A).$$

We call $\to$ and $\forall$ *computational*.
$\to$, $\forall$ and $\to^{\mathrm{nc}}$, $\forall^{\mathrm{nc}}$ are logically the same, but computationally different due to the realizability relation. Conjunction, disjunction and the existential quantifier are defined as inductive definitions.
In contrast to the BHK-interpretation we also consider concrete prime formulas, namely, inductively defined predicates.

$$t \mathbf{\ r\ } I\vec{s} := I^{\mathbf{r}}(t, \vec{s}).$$

where $I^{\mathbf{r}}$ is an inductive predicate, called a *witnessing predicate*, defined for each I.

Witnessing predicates

Consider the predicate $T_{\mathbf{L}}$ whose arity is $(\mathbf{L}_\alpha)$.

$$T_{\mathbf{L}}[], \qquad (T_{\mathbf{L}})_0^+$$

$$\forall^{\mathrm{nc}}_{x,u}(Qx \to T_{\mathbf{L}}\, u \to T_{\mathbf{L}}(x{::}u)). \qquad (T_{\mathbf{L}})_1^+$$

where Q is a predicate parameter of arity (α), an arbitrary type parameter.
The type of an inductive predicate I, namely, $\tau(I)$ is the algebra whose constructor types are the types of the introduction axioms.
Consider $T_{\mathbf{L}}$. By τ the introduction axioms go to the constructor types

$$\xi, \qquad \alpha \to \xi \to \xi,$$

which define the list algebra $\mathbf{L}_\alpha$.
We define the witnessing predicate $T^{\mathbf{r}}_{\mathbf{L}}$ of arity $(\tau(T_{\mathbf{L}}), \mathbf{L}_\alpha)$ as follows.

$$T^{\mathbf{r}}_{\mathbf{L}}([], []), \qquad (T^{\mathbf{r}}_{\mathbf{L}})_0^+$$

$$\forall^{\mathrm{nc}}_{x,y,u,v}(Q^*(y,x) \to T^{\mathbf{r}}_{\mathbf{L}}(v,u) \to T^{\mathbf{r}}_{\mathbf{L}}(y{::}v, x{::}u)). \qquad (T^{\mathbf{r}}_{\mathbf{L}})_1^+$$

where Q^* is a predicate parameter of arity $(\tau(Q), \alpha)$.

Program extraction

The notion of proof is given in natural deduction, which is represented in lambda terms. We define the program extraction et.

Definition (Program extraction)

Let M^A be a proof A. We define $\mathrm{et}(M^A)$ by induction on the construction of M^A.

$$\mathrm{et}(u^A) := x_{u^A}^{\tau(A)} \text{ where } x_{u^A} \text{ is uniquely associated with } u^A,$$

$$\mathrm{et}(I_i^+) := C_i, \qquad \mathrm{et}(I^-) := \mathcal{R}_\iota^\tau,$$

$$\mathrm{et}((\lambda_{u^A} M^B)^{A \to^{\mathrm{c}} B}) := \lambda_{x_u^{\tau(A)}}(\mathrm{et}(M)), \qquad \mathrm{et}((\lambda_{u^A} M^B)^{A \to^{\mathrm{nc}} B}) := \mathrm{et}(M),$$

$$\mathrm{et}(M^{A \to^{\mathrm{c}} B} N^A) := \mathrm{et}(M)\mathrm{et}(N), \qquad \mathrm{et}(M^{A \to^{\mathrm{nc}} B} N^A) := \mathrm{et}(M),$$

$$\mathrm{et}((\lambda_{x^\rho} M^A)^{\forall_x^{\mathrm{c}} A}) := \lambda_{x^\rho} \mathrm{et}(M), \qquad \mathrm{et}((\lambda_{x^\rho} M^A)^{\forall_x^{\mathrm{nc}} A}) := \mathrm{et}(M),$$

$$\mathrm{et}((M^{\forall_x^{\mathrm{c}} A} r)^{A(r)}) := \mathrm{et}(M) r, \qquad \mathrm{et}((M^{\forall_x^{\mathrm{nc}} A} r)^{A(r)}) := \mathrm{et}(M).$$

The following theorem claims that the program extraction finds a realizer.

Theorem (Soundness)

Let A be a formula and M be a proof of A under assumptions B_i for $i < k$. Then, there is a proof of $\mathrm{et}(M)\ \mathbf{r}\ A$ under the assumptions $u_i^{B_i}$ for $i < k$.

Branching trees

We consider arbitrarily branching trees based on the following *nested* algebra **Nt**.

$$\mathsf{Lf}^{\mathbf{Nt}}, \qquad \mathsf{Br}^{\mathbf{L}_{\mathbf{Nt}} \to \mathbf{Nt}}.$$

We can think about the combinations of the finiteness and the infiniteness.

- finite branching / finite height,
- infinite branching / finite height,
- finite branching / infinite height,
- infinite branching / infinite height.

We construct trees of finite branching / infinite height by using ${}^{\mathrm{co}}\mathcal{R}^{\rho}_{\mathbf{Nt}}$, the corecursion operator on **Nt**. The type of ${}^{\mathrm{co}}\mathcal{R}^{\rho}_{\mathbf{Nt}}$ and $\mathcal{R}^{\rho}_{\mathbf{Nt}}$ are:

$$\begin{aligned}
{}^{\mathrm{co}}\mathcal{R}^{\rho}_{\mathbf{Nt}} &: \rho \to (\rho \to \mathbf{U} + \mathbf{L}_{\mathbf{Nt}+\rho}) \to \mathbf{Nt}, \\
\mathcal{R}^{\rho}_{\mathbf{Nt}} &: \mathbf{Nt} \to \rho \to (\mathbf{L}_{\mathbf{Nt}\times\rho} \to \rho) \to \rho \\
&\approx \mathbf{Nt} \to (\mathbf{U} \to \rho) \to (\mathbf{L}_{\mathbf{Nt}\times\rho} \to \rho) \to \rho \\
&\approx \mathbf{Nt} \to (\mathbf{U} + \mathbf{L}_{\mathbf{Nt}\times\rho} \to \rho) \to \rho.
\end{aligned}$$

Corecursion operators

The outcome is determined by the result of applying the second argument to the first argument.

$$
\begin{aligned}
&{}^{\mathrm{co}}\mathcal{R}^{\rho}_{\mathbf{Nt}} : \rho \to (\rho \to \mathbf{U} + \mathbf{L}_{\mathbf{Nt}+\rho}) \to \mathbf{Nt},\\
&{}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{Nt}} \mapsto \lambda_{u,v}(\mathrm{Case}\ vu \text{ of } \mathsf{inl}\,() \to \mathsf{Lf}\\
&\qquad\qquad\qquad\qquad \mathsf{inr}\, x \to \mathsf{Br}(\mathcal{M}^{\mathbf{Nt}+\tau\to\mathbf{Nt}}_{\lambda_\alpha \mathbf{L}_\alpha} x[\mathrm{id}, \lambda_z({}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{Nt}} zv)])).
\end{aligned}
$$

where for $f^{\alpha\to\sigma}$ and $g^{\beta\to\sigma}$ we define $[f,g]^{\alpha+\beta\to\sigma}$ by

$$
[f,g](\mathsf{inl}\, x^{\alpha}) = f(x), \qquad [f,g](\mathsf{inr}\, y^{\beta}) = g(y).
$$

The map operator $\mathcal{M}$ constructs subtrees at each branch.

$$
\begin{aligned}
&\mathcal{M}^{\rho\to\sigma}_{\lambda_\alpha \mathbf{L}_\alpha} : \mathbf{L}_\rho \to (\rho \to \sigma) \to \mathbf{L}_\sigma,\\
&\mathcal{M}^{\rho\to\sigma}_{\lambda_\alpha \mathbf{L}_\alpha}\ []^{\rho}\, f = []^{\sigma},\\
&\mathcal{M}^{\rho\to\sigma}_{\lambda_\alpha \mathbf{L}_\alpha}\ (x{::}u)\, f = f(x){::}\mathcal{M}^{\rho\to\sigma}_{\lambda_\alpha \mathbf{L}_\alpha}\ u\, f.
\end{aligned}
$$

Destructors are given for each algebra as follows:

$$
\begin{aligned}
&\mathcal{D}_{\mathbf{Nt}} : \mathbf{Nt} \to \mathbf{U} + \mathbf{L}_{\mathbf{Nt}},\\
&\mathcal{D}_{\mathbf{Nt}}(\mathsf{Lf}) = \mathsf{inl}\,(), \qquad \mathcal{D}_{\mathbf{Nt}}(\mathsf{Br}\, u) = \mathsf{inr}\, u.
\end{aligned}
$$

Coinductive definitions

For an inductive predicate I we define its companion coinductive predicate ${}^{\mathrm{co}}I$. Let $T_{\mathbf{L}_\alpha}(Q)$ be a predicate stating a finite list of objects in Q.

$$T_{\mathbf{L}_\alpha}[], \qquad \forall^{\mathrm{nc}}_{x,u}(Qx \to T_{\mathbf{L}_\alpha} u \to T_{\mathbf{L}_\alpha}(x{::}u)).$$

Define $T_{\mathbf{Nt}}$ of arity $(\mathbf{Nt})$ to be:

$$T_{\mathbf{Nt}}(\mathrm{Lf}), \tag{$(T_{\mathbf{Nt}})_0^+$}$$

$$\forall^{\mathrm{nc}}_u(T_{\mathbf{L}_{\mathbf{Nt}}}(T_{\mathbf{Nt}})(u) \to T_{\mathbf{Nt}}(\mathrm{Br}u)). \tag{$(T_{\mathbf{Nt}})_1^+$}$$

The coinductive predicate ${}^{\mathrm{co}}T_{\mathbf{Nt}}$ of arity $(\mathbf{Nt})$ is defined by the *clause axiom* ${}^{\mathrm{co}}T_{\mathbf{Nt}}$, the dual of $(T_{\mathbf{Nt}})_0^+$ and $(T_{\mathbf{Nt}})_1^+$.

$$\forall^{\mathrm{nc}}_a({}^{\mathrm{co}}T_{\mathbf{Nt}}(a) \to a = \mathrm{Lf} \vee \exists_u(T_{\mathbf{L}_{\mathbf{Nt}}}({}^{\mathrm{co}}T_{\mathbf{Nt}})(u) \wedge a = \mathrm{Br}u)). \tag{$({}^{\mathrm{co}}T_{\mathbf{Nt}})^-$}$$

The *greatest-fixed-point axiom* (or *coinduction*) is given as follows:

$$\forall^{\mathrm{nc}}_a(Pa \to \forall^{\mathrm{nc}}_a(Pa \to a = \mathrm{Lf} \vee \exists_u(T_{\mathbf{L}_{\mathbf{Nt}}}({}^{\mathrm{co}}T_{\mathbf{Nt}} \vee P)(u) \wedge a = \mathrm{Br}u)) \to {}^{\mathrm{co}}T_{\mathbf{Nt}}(a)). \tag{$({}^{\mathrm{co}}T_{\mathbf{Nt}})^+$}$$

It states that ${}^{\mathrm{co}}T_{\mathbf{Nt}}$ is bigger than any competitor P that looks like ${}^{\mathrm{co}}T_{\mathbf{Nt}}$ in $({}^{\mathrm{co}}T_{\mathbf{Nt}})^-$. The realizability relation is extended to coinductive definitions. The program extraction is as well: $\mathrm{et}({}^{\mathrm{co}}I^-) := \mathcal{D}_{\tau(I)}$, $\mathrm{et}({}^{\mathrm{co}}I^+) := {}^{\mathrm{co}}\mathcal{R}^{\rho}_{\tau(I)}$.

Summary and remark on Minlog and its theory

1. Theory of computation
   - Free algebras as base types.
   - A term calculus with recursion, corecursion, general recursion, etc.
2. First order minimal logic (no $A \vee \neg A$) with inductive and coinductive definitions
   - Framework for constructive mathematics.
   - A language with $\to$, $\forall$, $\to^{\mathrm{nc}}$ and $\forall^{\mathrm{nc}}$.
   - Inductively and coinductively defined predicates can be introduced.
   - Support of classical proofs by A-translation and Dialectica interpretation.
3. Realizability interpretation
   - Provide the notion of *construction* in the BHK-interpretation.
   - Consider a relation $\mathbf{r}$ on a term t and a formula A, written as $t\ \mathbf{r}\ A$.
   - Intuitively means that t computationally solves the problem expressed by A.
     - Also possible to take $t\ \mathbf{r}\ A$ as a correctness notion.
   - We give a *program extraction* transforming a proof M of A into a realizer t of A.
     - The type of t is computed from A.
4. Minlog
   1. General purpose proof assistant.
   2. It has been developed for 20+ years in LMU Munich.
   3. We focus on a feature of program extraction.
   4. Download: `http://minlog-system.de/`.

Case study in exact real arithmetic

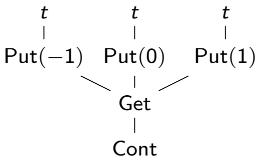
In the context of program extraction we study exact real arithmetic due to Ulrich Berger in Minlog. Consider two representations of uniform continuous functions in $[-1, 1]$:

- functional representation,
- infinite tree representation.

The latter one is done by corecursion in our setting.
Suppose we have the stream representation of real numbers.
Let **SD** be $-1, 0, 1$. Informally, a stream $\vec{d}$ of **SD** represents a real number $\sum_{i=0} \frac{d_i}{2^{i+1}}$.
The algebras of tree represented uniformly continuous functions are:

- $\mathbf{R}_\alpha$: Put of type $\mathbf{SD} \to \alpha \to \mathbf{R}$ and Get of type $\mathbf{R} \to \mathbf{R} \to \mathbf{R} \to \mathbf{R}$.
- $\mathbf{W}$: Stop of type $\mathbf{W}$ and Cont of type $\mathbf{R_W} \to \mathbf{W}$.

Define a term t ($\mathbf{R_W}$ finite, $\mathbf{W}$ infinite) to be:



This is the identity function $f(x) = x$.

A rational sequence $(a_n)_n$ is a Cauchy real if $\forall_k \exists_l \forall_{m,n \geqslant l}(|a_m - a_n| \leqslant 2^{-k})$.
These classical Cauchy reals are not suitable for computing, because we cannot find l in general.
We adopt a constructive version of Cauchy reals.

Definition (Cauchy reals)

A Cauchy real is given by a pair $\langle x^{\mathbf{N}\to\mathbf{Q}}, M^{\mathbf{N}\to\mathbf{N}} \rangle$ such that

$$\forall_k \forall_{m,n \geqslant M(k)}(|x(m) - x(n)| \leqslant 2^{-k}).$$

Based on a similar idea, we define uniformly continuous functions by a triple.

Definition (Uniformly continuous functions)

A uniformly continuous function is given by a triple $\langle h^{\mathbf{Q}\to\mathbf{N}\to\mathbf{N}}, \alpha^{\mathbf{N}\to\mathbf{N}}, \omega^{\mathbf{N}\to\mathbf{N}} \rangle$ (α is a Cauchy modulus, ω a modulus of uniform continuity) such that

$$\forall_k \forall_a \forall_{m,n \geqslant \alpha(k)}(|h(a,m) - h(a,n)| \leqslant 2^{-k}),$$

$$\forall_k \forall_{a,b} \forall_{n \geqslant \alpha(k)}(|a - b| \leqslant 2^{-\omega(k)+1} \to |h(a,n) - h(b,n)| \leqslant 2^{-k}).$$

Our running example

Let f be a uniformly continuous function in $[-1,1]$. We prove that the continuity of f, implies the productivity of f. We formulate

1. Abstract theory of uniformly continuous functions.
   - Good for simplicity if we don't want computational meaning from them.
   - Specify it by a type variable ϕ and axioms.
   - Make use of $\to^{\mathrm{nc}}$ and $\forall^{\mathrm{nc}}$.
2. Predicate C for the continuity.
   - $\mathbb{I}_{p,l} := [p-2^{-l}, p+2^{-l}]$, $B_{l,k} f := \forall_p \exists_q (f[\mathbb{I}_{p,l}] \subseteq \mathbb{I}_{q,k})$.
   - Let $C\,f$ be $\forall_k \exists_l B_{l,k} f$.
3. Predicate ${}^{\mathrm{co}}\mathrm{Write}$ for the productivity.
   - By a nested inductive conducive predicate.

Definition (Inductive predicate Read_X and coinductive predicate ${}^{\mathrm{co}}\mathrm{Write}$)

Let X be a predicate variable of arity ϕ. Also let $(\mathrm{Out}_d \circ f)(x)$ be $2f(x) - d$ and $(f \circ \mathrm{In}_d)(x)$ be $f(\frac{x+d}{2})$.

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

$$\forall^{\mathrm{nc}}_f (\mathrm{Read}_X(f \circ \mathrm{In}_{-1}) \to \mathrm{Read}_X(f \circ \mathrm{In}_0) \to \mathrm{Read}_X(f \circ \mathrm{In}_1) \to \mathrm{Read}_X f), \quad (\mathrm{Read})^+_1$$

$$\forall^{\mathrm{nc}}_f ({}^{\mathrm{co}}\mathrm{Write} f \to f = \mathrm{Id} \vee \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}} f). \qquad ({}^{\mathrm{co}}\mathrm{Write})^-$$

Continuity to productivity

Proposition (Continuity to productivity)

$\forall^{\mathrm{nc}}_f (C\, f \to {}^{\mathrm{co}}\mathrm{Write} f)$.

Proof.

Let f be given and assume $C\, f$. Use the greatest-fixed-point axiom for ${}^{\mathrm{co}}\mathrm{Write}\, f$. We instantiate the competitor predicate P by C as follows.

$$\forall^{\mathrm{nc}}_f (C\, f \to \forall^{\mathrm{nc}}_f (C\, f \to f = \mathrm{Id} \lor \mathrm{Read}_{C \lor {}^{\mathrm{co}}\mathrm{Write}} f) \to {}^{\mathrm{co}}\mathrm{Write} f).$$

It suffices to prove the second premise of the above formula. Let f be given and assume $C\, f$. Since $C\, f$ is same as $\forall_k \exists_l B_{l,k} f$, it implies $\exists_l B_{l,2} f$. We prove the right disjunct by the following lemma. □

Lemma

$\forall_l \forall^{\mathrm{nc}}_f (B_{l,2} f \to C\, f \to \mathrm{Read}_{C \lor {}^{\mathrm{co}}\mathrm{Write}} f)$.

Proof.

By induction on l. □

Extracted program

Let M be our proof of Proposition. By program extraction, we get $\mathrm{et}(M)$ as a realizer of $\forall^{\mathrm{nc}}_f(C\,f \to {}^{\mathrm{co}}\mathrm{Write} f)$.
The extracted program $t := \mathrm{et}(M)$ is of type $(\mathbf{N} \to \mathbf{N} \times (\mathbf{Q} \to \mathbf{Q})) \to \mathbf{W}$ where $\mathbf{R}_\alpha$ and $\mathbf{W}$ are computed from ${}^{\mathrm{co}}\mathrm{Write}$ and Read_X.
For a given uniformly continuous function $\langle h, \alpha, \omega \rangle$, $\mathtt{t}$ computes a non-well founded tree representing $\langle h, \alpha, \omega \rangle$.
Defining $f(x) = -x$ by h, α and ω, $\mathtt{t}(\lambda_n \langle \omega(n), \lambda_a h(a, \alpha(n)) \rangle)$ gives the following tree.

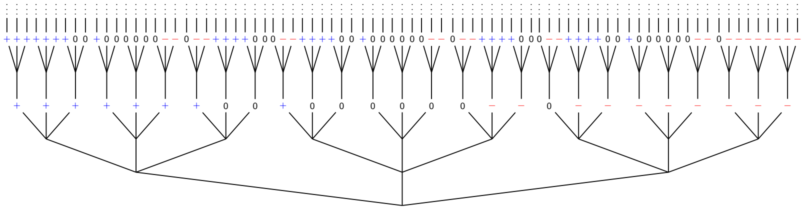


Figure : Type-0 representation of $f(x) = -x$.

In the figure $-$, 0 and $+$ stands for -1, 0 and 1, respectively.

Concluding remarks

- Related work
  - Program extraction from coinductive definitions by Tatsuta (1998).
  - Program extraction from coind. defs. in typed setting by Berger (2009).
  - Theory of computable functionals (the theory of Minlog) by S & Wainer (2012).
  - Proof assistants: Coq, Isabelle, Nuprl, Agda, Matita, and so on.
- Case studies in exact real arithmetic running in Minlog
  - Two representations of u.c.functions, application, composition and integration by M.
  - Intermediate value theorem by S (2008, in functional representation).
  - ODE solver from Picard-Lindelöf Thm. by Thilo Weghorn (2013, in fun. rep.).
  - `http://www.minlog-system.de/`.