Interactive Proofs in Dependent Type Theory

Anton Setzer
(Joint work with Peter Hancock)

1. Definition of the IO Monad in type theory.
2. Run, redirection and equality.
3. Well-founded version.
4. State-dependent IO.

1. Definition of the IO Monad in Type Theory

Direction in Functional Programming

Design of programming languages based on dependent types.

Theoretical Problems:

- Equality. Hard.
- Practical structuring of programs.
  - ∗ Local variables.
  - ∗ Record types.

  Unproblematic.
- Polymorphism, subtyping.
- Input/output.

Main models for input/output:

- Streams.

  Timing between input/output depends on evaluation strategy.

  Only fixed finite number of IO-devices.
- The IO-monad.

Monad

A monad is a tripel $(M, \eta, *)$, where
- M : Set $\to$ Set,
- η : $(A$: Set$, a$: $A) \to M(A)$,
- $*$: $(A$: Set$, B$: Set$, p$: $M(A), q$: $A \to M(B))$,
    $\to M(B)$,

with abbreviations
$\eta_a := \eta_a^A := \eta(A, a)$,
$p * q := p *_{A,B} q := *(A, B, p, q)$,

s.t. for A, B, C : Set$, a$: A, p : $M(A)$,
    q : $A \to M(B), r$: $B \to M(C)$:
- $\eta_a * q = q(a)$.
- $p * (x)\eta_x = p$.
- $(p * q) * r = p * (x)(q(x) * r)$.

IO-Monad

IO-Monad = monad $(\mathrm{IO}, \eta, *)$ with interpretation:

- $\mathrm{IO}(A)$ = set of interactive programs which, if terminating, returns an element $a : A$.
- η_a = program with no interaction, returns a.
- $*$ = composition of programs.

Additional operations added like
input$(d, A) : \mathrm{IO}(A)$
    input from device d an element $a : A$
    and return a.
output$(d, A) : A \to \mathrm{IO}(1)$
    for $a : A$ output a on device d
    and return $<> : 1$.

IO-Monad in Haskell:
Small part of the program interactive.
Large part purely functional.

Problems of the IO-Monad:

- $*$ cannot be a constructor.
- Equalities can hold only extensionally.

The IO-tree

A world w is a pair (C, R) s.t.
- C : Set (Commands).
- $R : C \to$ Set (responses to a command).

Assume $w = (C, R)$ a world.

$\mathrm{IO}_w(A)$ or shorter $\mathrm{IO}(A)$ is the set of (possibly non-wellfounded) trees with
- leaves in A.
- nodes marked with elements of C.
- nodes marked with c have branching degree $R(c)$.

$$\frac{A : \mathsf{Set}}{\mathrm{IO}_w(A) : \mathsf{Set}}$$

$$\frac{a : A}{\mathsf{leaf}(a) : \mathrm{IO}_w(A)}$$

$$\frac{c : C \qquad\qquad p : R(c) \to \mathrm{IO}_w(A)}{\mathsf{do}(c, p) : \mathrm{IO}_w(A)}$$

Note: $\mathrm{IO}_w(A)$ now parametrized w.r.t. w.

New function execute**:**

Status:
- Like function "normalize".

- No construction inside type theory.

Let w_0 be a fixed world (real commands).

execute takes an element $p : \mathrm{IO}_{w_0}(A)$ and does the following:
- It reduces p to canonical form.
- If $p = \mathrm{leaf}(a)$ it terminates and returns a.
- If $p = \mathrm{do}(c, q)$, then it
    - carries out command c;
    - interprets the result as an element $r : R(c)$;
    - then continues with $q(r)$.

Essentially normalization p but with interaction with the real world.

Definition of η, $*$

$\eta_a = \mathsf{leaf}(a)$.
$\mathsf{leaf}(a) * q = q(a)$.
$\mathsf{do}(c, p) * q = \mathsf{do}(c, (x)(p(x) * q))$.

For well-founded trees monad laws provable w.r.t. extensional equality.

Additional function carryout**:**
$\mathsf{carryout} : (c : C) \to \mathrm{IO}(R(c))$.
$\mathsf{carryout}(c) = \mathsf{do}(c, (x)\mathsf{leaf}(x))$.

2. Run, Redirect, Equality

2.1. Run

Problem: Interactive programs should not terminate after finite amount of time.
Run-construction:

Works only for trees which are not leaves.

$$\frac{A : \mathrm{Set}}{\mathrm{IO}^{+}(A) : \mathrm{Set}} \qquad \frac{a : \mathrm{IO}^{+}(A)}{a^{-} : \mathrm{IO}(A)}$$

$$\frac{c : C \qquad p : R(c) \to \mathrm{IO}(A)}{\mathrm{do}^{+}(c,p) : \mathrm{IO}^{+}(A)}$$

$$\mathrm{do}^{+}(c,p)^{-} = \mathrm{do}(c,p)$$

Assume A, B : Set.

$$\frac{b : B \qquad\qquad q : B \to \mathrm{IO}^{+}(A+B)}{\mathsf{run}(b,q) : \mathrm{IO}(A)}$$

Auxiliary function run$'$ needed

$$\frac{p : \mathrm{IO}(A+B) \qquad\qquad q : B \to \mathrm{IO}^{+}(A+B)}{\mathsf{run}'(p,q) : \mathrm{IO}(A)}$$

$\mathsf{run}(b,q) = \mathsf{run}'(\mathsf{leaf}(b), q)$
$\mathsf{run}'(\mathsf{leaf}(i(a)), q) = \mathsf{leaf}(a)$
$\mathsf{run}'(\mathsf{leaf}(j(b)), q) = \mathsf{run}'(q(b)^{-}, q)$
$\mathsf{run}'(\mathsf{do}(c,p), q) = \mathsf{do}(c, (x)\mathsf{run}'(p(x), q))$

Remark We can define run s.t.
$\mathsf{run}(b,q) : \mathrm{IO}^{+}(B)$.

2.2. Redirect

Assume

- $w = (C, R)$, $w' = (C', R')$ are worlds.
- A : Set,
- $p : \mathrm{IO}_w(A)$.
- $q : (c : C) \to \mathrm{IO}^{+}_{w'}(R(c))$.

Define redirect$(p, q) : \mathrm{IO}_{w'} A$:

redirect(leaf$(a), q) =$ leaf(a).
redirect(do$(c, p), q) = q(c)^{-}*(x)$redirect$(p(x), q)$.

2.3. Equality

Equality corresponding to extensional equality on non-wellfounded trees:
Bisimulation:

$$\frac{p : \mathrm{IO}(A) \qquad q : \mathrm{IO}(A)}{\mathsf{Eq}(p, q) : \mathsf{Set}}$$

$$\frac{p : \mathrm{IO}(A) \qquad q : \mathrm{IO}(A) \qquad n : \mathsf{N}}{\mathsf{Eq}'(n, p, q) : \mathsf{Set}}$$

$\mathsf{Eq}(p, q) = \forall n : \mathsf{N}.\mathsf{Eq}'(n, p, q)$.

$\mathsf{Eq}'({}^{0}_{\mathsf{S}(n)}, \mathsf{leaf}(a), \mathsf{do}(c, p))$
$= \mathsf{Eq}'({}^{0}_{\mathsf{S}(n)}, \mathsf{do}(c, p), \mathsf{leaf}(a)) = \bot$

$\mathsf{Eq}'(0, \mathsf{leaf}(a), \mathsf{leaf}(a')) = \mathrm{I}(A, a, a')$.
$\mathsf{Eq}'(0, \mathsf{do}(c, p), \mathsf{do}(c', p')) = \mathrm{I}(C, c, c')$.

$\mathsf{Eq}'(\mathsf{S}(n), \mathsf{leaf}(a), \mathsf{leaf}(a')) = \mathrm{I}(A, a, a')$.
$\mathsf{Eq}'(\mathsf{S}(n), \mathsf{do}(c, p), \mathsf{do}(c', p')) =$
$\quad \Sigma q : \mathrm{I}(C, c, c').\forall r : R(c).\mathsf{Eq}(n, p(r), p'(\cdots r \cdots))$.

- Eq seems to be the natural extension of extensional equality to non-well-founded trees (but then I has to be replaced by extensional equality).

- Monad laws w.r.t. Eq are provable.

- Two programs are equal w.r.t. Eq, if their IO-behaviour is identical.
  $\Rightarrow$ Extensionally, for every IO-behaviour there is exactly one program.
  $\Rightarrow$ IO-tree = suitable model of IO.

Problem: non-normalizing

Let $A = C = \mathsf{N}$, $\mathsf{R}(c)$ arbiterary.
Assume $f : \mathsf{N} \to \mathsf{N}$.
$p := (n)\mathsf{do}^{+}(f(n), (x)\mathsf{leaf}(n{+}1)) : \mathsf{N} \to \mathrm{IO}^{+}(A)$.

$$
\begin{array}{ll}
\multicolumn{2}{l}{\mathsf{run}(0, p)} \\
\longrightarrow & \mathsf{run}'(p(0)^{-}, p) \\
\longrightarrow & \mathsf{do}(f(0), (x)\mathsf{run}'(\mathsf{leaf}(1), p)) \\
\longrightarrow & \mathsf{do}(f(0), (x)\mathsf{run}'(p(1)^{-}, p)) \\
\longrightarrow & \mathsf{do}(f(0), (x)\mathsf{do}(f(1), (y)\mathsf{run}'(\mathsf{leaf}(2), (z)p))) \\
\longrightarrow & \cdots \\
\longrightarrow & \mathsf{do}(f(0), (x)\mathsf{do}(f(1), (y)\mathsf{do}(f(2), (z) \cdots)))
\end{array}
$$

Consequence: with intensional equality type-checking undecidable.

Two ways to remedy this:

1) Consider a restriction of the above s.t.

- Non-well-founded objects are only reduced to canonical form.
- No intensional equality on non-well-founded objects.
- Develop suitable elimination rules.

Difficult, but challenging.

2) Represent non-wellfounded trees by well-founded ones.

3. Well-founded version

Add run as a constructor.

Problem: run refers to $\mathrm{IO}(A + B)$.
Therefore run needs to be defined simultaneously for all sets.

Restrict the above to a universe.

Assume
$\mathsf{U} : \mathsf{Set}$, $\mathsf{T} : \mathsf{U} \to \mathsf{Set}$.
$\widehat{+} : U \to U \to \mathsf{Set}$, $\mathsf{T}(\hat{A}\widehat{+}\hat{B}) = \mathsf{T}(\hat{A}) + \mathsf{T}(\hat{B})$.
Assume $w = (C, R)$ is a world.

For $\hat{A} : \mathsf{U}$ let $A := \mathsf{T}(\hat{A})$ similarly for $\hat{B}$, $\hat{C}$.

$$\frac{\widehat{A} : \mathsf{U}}{\mathsf{IO}_w(\widehat{A}) : \mathsf{Set}} \qquad \frac{\widehat{A} : \mathsf{U}}{\mathsf{IO}^+_w(\widehat{A}) : \mathsf{Set}}$$

$$\frac{p : \mathsf{IO}^+(\widehat{A})}{p^- : \mathsf{IO}(\widehat{A})}$$

$$\frac{a : A}{\mathsf{leaf}(a) : \mathsf{IO}(\widehat{A})}$$

$$\frac{c : C \qquad p : R(c) \to \mathsf{IO}(\widehat{A})}{\mathsf{do}^{(+)}(c, p) : \mathsf{IO}^{(+)}(\widehat{A})}$$

$$\mathsf{do}^+(c, p)^- = \mathsf{do}(c, p)$$

$$\frac{\widehat{B} : \mathsf{U} \qquad b : B \qquad p : B \to \mathsf{IO}^+(\widehat{A} \widehat{+} \widehat{B})}{\mathsf{run}^{(+)}(\widehat{B}, b, p) : \mathsf{IO}^{(+)}(\widehat{A})}$$

$$\mathsf{run}^+(\widehat{B}, b, p)^- = \mathsf{run}(\widehat{B}, b, p).$$

Let $\mathrm{IO}^{\mathrm{wf},(+)}(A)$ be the set $\mathrm{IO}^{(+)}(A)$ as defined in this section.

Let $\mathrm{IO}^{\mathrm{nonwf}(+)}(A)$ be $\mathrm{IO}^{(+)}(A)$ as defined before.

Define $\mathrm{emb}^{(+)}_{\widehat{A}} : \mathrm{IO}^{\mathrm{wf},(+)}(\widehat{A}) \to \mathrm{IO}^{\mathrm{nonwf},(+)}(A)$:

$\mathrm{emb}(\mathrm{leaf}(a)) = \mathrm{leaf}(a)$.
$\mathrm{emb}^{(+)}(\mathrm{do}^{(+)}(c,p)) = \mathrm{do}^{(+)}(c,(x)\mathrm{emb}(p(x)))$.
$\mathrm{emb}^{(+)}(\mathrm{run}^{(+)}(\widehat{B},b,p)) =$
  $\mathrm{run}^{(+)}(B,b,(x)\mathrm{emb}^{+}_{\widehat{A}\widehat{+}\widehat{B}}(p(x)))$

Now η, $*$, redirect, Eq on $\mathrm{IO}^{\mathrm{nonwf}}_{w}(A)$ can be mimiced by corresponding operations on $\mathrm{IO}^{\mathrm{wf}}_{w}(A)$.

execute **on** $\mathrm{IO}^{\mathsf{wf}}$

Define
decompose : $\mathrm{IO}^{\mathsf{wf}}(A) \to$

$$A + \Sigma c : C.(R(c) \to \mathrm{IO}^{\mathsf{wf}}(A))$$

which corresponds to the decomposition of an element in $\mathrm{IO}^{\mathsf{nonwf}}(A)$ into the arguments of its constructor.

execute(p) does now the following:

- If decompose(p) $= i(a)$, then terminate with result a.
- If decompose(p) $= j(<c, q>)$, then carry out command c, get response r and continue with $q(r)$.

Result:

- All derivable terms are strongly normalizing.

- Therefore in the beginning and after every IO-command execute will terminate either completely or carry out the next IO-command.

- However, execute might carry out infinitely many IO-commands.

- Notion of “strongly-normalizing IO-programs”.

4. State-dependent IO

For simplicity we will work with non-well-founded trees.

Now let set of commands be influenced by commands, e.g.
- open a new window.
- switch on printer.

A world is now a quadrupel (S, C, R, ns) s.t.
- S : Set (set of states).
- $C : S \to$ Set (set of commands).
- $R : (s : S, C(s)) \to$ Set (set of responses).
- $ns : (s : S, c : C(s), r : R(c, s)) \to S$
  (next state).

Let $w = (S, C, R, ns)$ be a world.

$$\frac{OP : S \to \mathsf{Set} \qquad\qquad s : S}{\mathsf{tree}(OP, s) : \mathsf{Set}}$$

Assume $OP : S \to \mathsf{Set}$.

$$\frac{s : S \qquad\qquad p : OP(s)}{\mathsf{leaf}(p) : \mathsf{tree}(OP, s)}$$

$$\frac{\begin{array}{c} s : S \\ c : C(s) \\ p : (r : R(s, c) \to \mathsf{tree}(OP, ns(s, c, r))) \end{array}}{\mathsf{do}(c, p) : \mathsf{tree}(OP, s)}$$

$$\frac{IP : S \to \mathsf{Set} \qquad\qquad OP : S \to \mathsf{Set}}{\mathsf{IO}(IP, OP) : \mathsf{Set}}$$

$$\mathsf{IO}(IP, OP) = \Pi s : S.(IP(s) \to \mathsf{tree}(OP, s)).$$

We can now define:

$\eta_{IP} : \mathrm{IO}(IP, IP)$.

$*_{IP,OP_0,OP_1}$:
$\mathrm{IO}(IP, OP_0) \to \mathrm{IO}(OP_0, OP_1) \to \mathrm{IO}(IP, OP_1)$.

run_{OP_0,OP_1} :
$\mathrm{IO}(OP_0, (s)(OP_0(s)+OP_1(s))) \to \mathrm{IO}(OP_0, OP_1)$.

Redirection

Define $\mathrm{IO}^{+}_{w}(IP, OP)$ as before.

Assume

- $w = (S, C, R, ns)$, $w' = (S', C', R', ns')$ are worlds.
- $Rel : S \to S' \to$ Set,
- $IP, OP : S \to$ Set,
- $p : (s : S, c : C(s))$
  $\to \mathrm{IO}^{+}_{w'}((s')Rel(s, s'),$
  $(s')\Sigma r : R(s, c).Rel(ns(s, c, r), s'))$

Define

redirect(Rel, IP, OP, p) :

$\mathrm{IO}_{w}(IP, OP) \to$

$\mathrm{IO}_{w'}((s')\Sigma s : S.(Rel(s, s') \wedge IP(s)),$

$(s')\Sigma s : S.(Rel(s, s') \wedge OP(s)))$.

execute

Let $w_0 = (S_0, C_0, R_0, ns_0)$ be a standard world, $s_0 : S$ be a state the system is always in (state of unknowing).
Assume $p : \mathrm{tree}_{w_0}(OP, s_0)$.

execute applied to p normalizes p by carrying out commands as before.