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.
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:
   - 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 triple \((M, \eta, \ast)\), where

- \(M : \text{Set} \rightarrow \text{Set}\),
- \(\eta : (A : \text{Set}, a : A) \rightarrow M(A)\),
- \(\ast : (A : \text{Set}, B : \text{Set}, p : M(A), q : A \rightarrow M(B))\),
  \(\rightarrow M(B)\),

with abbreviations

\[\eta_a := \eta^A_a := \eta(A, a),\]
\[p \ast q := p \ast_{A, B} q := \ast(A, B, p, q),\]

s.t. for \(A, B, C : \text{Set}, a : A, p : M(A)\),
\(q : A \rightarrow M(B), r : B \rightarrow M(C)\):
- \(\eta_a \ast q = q(a)\).
- \(p \ast (x) \eta_x = p\).
- \((p \ast q) \ast r = p \ast (x) (q(x) \ast r)\).
**IO-Monad**

IO-Monad = monad \((\text{IO}, \eta, \ast)\) with interpretation:

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

Additional operations added like

\[
\begin{align*}
\text{input}(d, A) : \text{IO}(A) & \quad \text{input from device } d \text{ an element } a : A \text{ and return } a. \\
\text{output}(d, A) : A \rightarrow \text{IO}(1) & \quad \text{for } a : A \text{ output } a \text{ on device } d \text{ and return } \langle \rangle : 1.
\end{align*}
\]

**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.
- $C : \text{Set (Commands)}$.
- $R : C \to \text{Set (responses to a command)}$.

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

$\text{IO}_w(A)$ or shorter $\text{IO}(A)$ 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 $R(c)$.
\[ A : \text{Set} \]
\[ \text{IO}_w(A) : \text{Set} \]

\[ a : A \]
\[ \text{leaf}(a) : \text{IO}_w(A) \]

\[ c : C \quad p : R(c) \rightarrow \text{IO}_w(A) \]
\[ \text{do}(c, p) : \text{IO}_w(A) \]

**Note:** \( \text{IO}_w(A) \) 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 $p : \text{IO}_{w_0}(A)$ and does the following:
- It reduces $p$ to canonical form.
- If $p = \text{leaf}(a)$ it terminates and returns $a$.
- If $p = \text{do}(c, q)$, 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 $\eta$, *

\[ \eta_a = \text{leaf}(a). \]
\[ \text{leaf}(a) * q = q(a). \]
\[ \text{do}(c, p) * q = \text{do}(c, (x)(p(x) * q)). \]

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

**Additional function** carryout:
\[ \text{carryout} : (c : C) \rightarrow \text{IO}(R(c)). \]
\[ \text{carryout}(c) = \text{do}(c, (x)\text{leaf}(x)). \]
2. Run, Redirect, Equality

2.1. Run

**Problem:** Interactive programs should not terminate after finite amount of time.

Run-construction:

\[
\begin{align*}
A : \text{Set} & \quad & a : \text{IO}^+(A) \\
\text{IO}^+(A) : \text{Set} & \quad & a^- : \text{IO}(A) \\
c : C & \quad & p : R(c) \to \text{IO}(A) \\
\text{do}^+(c, p) : \text{IO}^+(A) \\
\text{do}^+(c, p)^- = \text{do}(c, p)
\end{align*}
\]

Works only for trees which are not leaves.
Assume $A, B : \text{Set}$.

\[
\begin{align*}
  b : B & \quad q : B \rightarrow \text{IO}^+(A + B) \\
  \text{run}(b, q) : \text{IO}(A)
\end{align*}
\]

Auxiliary function run' needed

\[
\begin{align*}
  p : \text{IO}(A + B) & \quad q : B \rightarrow \text{IO}^+(A + B) \\
  \text{run}'(p, q) : \text{IO}(A)
\end{align*}
\]

\[
\begin{align*}
  \text{run}(b, q) &= \text{run}'(\text{leaf}(b), q) \\
  \text{run}'(\text{leaf}(i(a)), q) &= \text{leaf}(a) \\
  \text{run}'(\text{leaf}(j(b)), q) &= \text{run}'(q(b)^-, q) \\
  \text{run}'(\text{do}(c, p), q) &= \text{do}(c, (x) \text{run}'(p(x), q))
\end{align*}
\]

Remark We can define run s.t. \[ \text{run}(b, q) : \text{IO}^+(B). \]
2.2. Redirect

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

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

\[
\begin{align*}
\text{redirect}(&\text{leaf}(a), q) = \text{leaf}(a) , \\
\text{redirect}(&\text{do}(c, p), q) = q(c)\text{^-*}(x)\text{redirect}(p(x), q).
\end{align*}
\]
2.3. Equality

Equality corresponding to extensional equality on non-wellfounded trees:

Bisimulation:

\[
p : \text{IO}(A) \quad q : \text{IO}(A) \\
\text{Eq}(p, q) : \text{Set}
\]

\[
p : \text{IO}(A) \quad q : \text{IO}(A) \quad n : \mathbb{N} \\
\text{Eq}^{'(n, p, q)} : \text{Set}
\]

\[
\text{Eq}(p, q) = \forall n : \mathbb{N}. \text{Eq}^{'(n, p, q)}.
\]

\[
\text{Eq}^{'(S(n), \text{leaf}(a), \text{do}(c, p))} \\
= \text{Eq}^{'(S(n), \text{do}(c, p), \text{leaf}(a))} = \perp
\]

\[
\text{Eq}^{'(0, \text{leaf}(a), \text{leaf}(a'))} = I(A, a, a'). \\
\text{Eq}^{'(0, \text{do}(c, p), \text{do}(c', p'))} = I(C, c, c').
\]

\[
\text{Eq}^{'(S(n), \text{leaf}(a), \text{leaf}(a'))} = I(A, a, a'). \\
\text{Eq}^{'(S(n), \text{do}(c, p), \text{do}(c', p'))} = \\
\sum q : I(C, c, c'). \forall r : R(c). \text{Eq}(n, p(r), p'(\cdots r \cdots)).
\]
- 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.
  \[ \Rightarrow \text{Extensionally, for every IO-behaviour there is exactly one program.} \]
  \[ \Rightarrow \text{IO-tree } = \text{ suitable model of IO.} \]
Problem: non-normalizing

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

run(0, p)
$\quad\rightarrow\quad$ run$'(p(0)^-, p)$
$\quad\rightarrow\quad$ do$(f(0), (x)\text{run}'(\text{leaf}(1), p))$
$\quad\rightarrow\quad$ do$(f(0), (x)\text{run}'(p(1)^-, p))$
$\quad\rightarrow\quad$ do$(f(0), (x)\text{do}(f(1), (y)\text{run}'(\text{leaf}(2), (z)p)))$
$\quad\rightarrow\quad$ \ldots
$\quad\rightarrow\quad$ do$(f(0), (x)\text{do}(f(1), (y)\text{do}(f(2), (z)\ldots))))$

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 \( \text{IO}(A + B) \).
Therefore run needs to be defined simultaneously for all sets.

Restrict the above to a universe.

Assume
\[
U : \text{Set}, \quad T : U \to \text{Set}.
\]
\[
\widehat{+} : U \to U \to \text{Set}, \quad T(\widehat{A} + \widehat{B}) = T(\widehat{A}) + T(\widehat{B}).
\]
Assume \( w = (C, R) \) is a world.

For \( \widehat{A} : U \) let \( A := T(\widehat{A}) \) similarly for \( \widehat{B}, \widehat{C} \).
\[
\begin{array}{c}
\text{\(\hat{A} : U\)} \\
\text{\(\IO_w(\hat{A}) : \text{Set}\)}
\end{array}
\quad
\begin{array}{c}
\text{\(\hat{A} : U\)} \\
\text{\(\IO_w^+(\hat{A}) : \text{Set}\)}
\end{array}
\]

\[
p : \IO^+(\hat{A}) \\
p^- : \IO(\hat{A})
\]

\[
a : A \\
\text{leaf}(a) : \IO(\hat{A})
\]

\[
c : C \\
p : R(c) \to \IO(\hat{A})
\]

\[
do^+(c, p) : \IO^+(\hat{A})
\]

\[
do^+(c, p)^- = do(c, p)
\]

\[
\hat{B} : U \\
b : B \\
p : B \to \IO^+(\hat{A} \hat{+} \hat{B})
\]

\[
\text{run}^+(\hat{B}, b, p) : \IO^+(\hat{A})
\]

\[
\text{run}^+(\hat{B}, b, p)^- = \text{run}(\hat{B}, b, p).
\]
Let $\mathcal{IO}^{\text{wf},(+)}(A)$ be the set $\mathcal{IO}^{(+)}(A)$ as defined in this section.

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

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

\begin{align*}
\text{emb}(\text{leaf}(a)) &= \text{leaf}(a). \\
\text{emb}(+) (\text{do}(+)(c, p)) &= \text{do}(+)(c, (x)\text{emb}(p(x))). \\
\text{emb}(+) (\text{run}(+)(\widehat{B}, b, p)) &= \\
&\text{run}(+)(B, b, (x)\text{emb}^{+}_{\widehat{A} + \widehat{B}}(p(x)))
\end{align*}

Now $\eta$, $\ast$, redirect, Eq on $\mathcal{IO}^{\text{nonwf}}_{\text{w}}(A)$ can be mimicked by corresponding operations on $\mathcal{IO}^{\text{wf}}_{\text{w}}(A)$. 

19
execute on $\text{IO}^{\text{wf}}$

Define
$\text{decompose} : \text{IO}^{\text{wf}}(A) \rightarrow$

$A + \sum c : C.(R(c) \rightarrow \text{IO}^{\text{wf}}(A))$

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

$\text{execute}(p)$ does now the following:

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

20
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.
- \(S : \text{Set} \) (set of states).
- \(C : S \rightarrow \text{Set} \) (set of commands).
- \(R : (s : S, C(s)) \rightarrow \text{Set} \) (set of responses).
- \(ns : (s : S, c : C(s), r : R(c, s)) \rightarrow S \) (next state).

Let \(w = (S, C, R, ns)\) be a world.
\[ \text{Assume } OP : S \rightarrow \text{Set.} \]

\[ \begin{array}{c}
  s : S \\
  p : OP(s) \\
  \text{leaf}(p) : \text{tree}(OP, s)
\end{array} \]

\[ \begin{array}{c}
  s : S \\
  c : C(s) \\
  p : (r : R(s, c) \rightarrow \text{tree}(OP, ns(s, c, r))) \\
  \text{do}(c, p) : \text{tree}(OP, s)
\end{array} \]

\[ \begin{array}{c}
  IP : S \rightarrow \text{Set} \\
  OP : S \rightarrow \text{Set}
\end{array} \]

\[ \text{IO}(IP, OP) : \text{Set} \]

\[ \text{IO}(IP, OP) = \prod s : S. (IP(s) \rightarrow \text{tree}(OP, s)). \]
We can now define:

\( \eta_{IP} : \text{IO}(IP, IP) \).

\[*_{IP, OP_0, OP_1} : \]
\( \text{IO}(IP, OP_0) \rightarrow \text{IO}(OP_0, OP_1) \rightarrow \text{IO}(IP, OP_1) \).

\( \text{run}_{OP_0, OP_1} : \)
\( \text{IO}(OP_0, (s)(OP_0(s) + OP_1(s))) \rightarrow \text{IO}(OP_0, OP_1) \).
Redirection

Define \( \text{IO}^+_w(IP, OP) \) as before.

Assume

- \( w = (S, C, R, ns) \), \( w' = (S', C', R', ns') \) are worlds.
- \( Rel : S \rightarrow S' \rightarrow \text{Set} \),
- \( IP, OP : S \rightarrow \text{Set} \),
- \( p : (s : S, c : C(s)) \)
  \[ \rightarrow \text{IO}^+_w((s')Rel(s, s'), \]
  \[ (s')\sum r : R(s, c).Rel(ns(s, c, r), s')) \]

Define

\( \text{redirect}(Rel, IP, OP, p) : \)
\[ \text{IO}_w(IP, OP) \rightarrow \]
\[ \text{IO}_{w'}((s')\sum s : S.(Rel(s, s') \land IP(s)), \]
\[ (s')\sum s : S.(Rel(s, s') \land 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 unkowning).
Assume $p : \text{tree}_{w_0}(OP, s_0)$.

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