The IO Monad in Dependent Type Theory

Peter Hancock, Edinburgh, Scotland
pgh@dcs.ed.ac.uk
Anton Setzer, Uppsala, Sweden
setzer@math.uu.se

April 13, 1999

1. Definition of the IO Monad in type theory.
2. While, redirect and equality.
3. Normalizing version.
4. State-dependent IO.
5. Parallelism, Nondeterminism.

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.

Models for input/output:
- Streams.
  Difficulties with infinitely many input/output devices
  Timing between input/output depends on evaluation strategy.
- The IO-monad.

Monad

A monad is a triple $(M, \eta, *)$, 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 : $A \to M(B), r$: $B \to M(C)$:
- $\eta_a * q = q(a)$.
- $p * \lambda x.\eta_x = p$.
- $(p * q) * r = p * \lambda x.(q(x) * r)$.

IO-Monad

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

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

Additional elements 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.
  $\Rightarrow$ Monads do not fit into the conceptual framework of Martin-Löf type theory.

- 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-well-founded) trees with
- leaves in A.
- nodes marked with elements of C.
- nodes marked with c have branching degree $R(c)$.

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

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

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

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

New operation execute**:**

Status:
- Like function "reduce to canonical form".

- No construction inside type theory.

Let w_0 be a fixed world (real commands).

execute applied to $p : \mathrm{IO}_{w_0}(A)$ 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 of p but with interaction with the real world.

Definition of η, $*$

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

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

Additional function interact**:**

$\text{interact} : (c : C) \to \text{IO}(R(c))$.
$\text{interact}(c) = \text{do}(c, \lambda x.\text{leaf}(x))$.

interact(c) executes command c and returns the result.

2. While, Redirect, Equality

2.1. While

Problem:

- Interactive programs should possibly have infinitely many interactions (no termination after finite amount of time).

Add while-loop:

Assume:
- a set B
- an initial value $b : B$
- $q : B \to (\mathrm{IO}(A) + \mathrm{IO}(B))$.

while$_B(b, q) : \mathrm{IO}(A)$ does the following:
- If $q(b)$ is in $\mathrm{IO}(B)$ then it runs this program.
  If it terminates with leaf b', it continues with while$_B(b', q)$.
- If $q(b)$ is in $\mathrm{IO}(A)$ then it run this program.
  When it stops it returns the result.

Problem:
Black hole recursion for trees which consist of leaves.

Therefore define set of trees which have at least one command at the root:

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

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

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

Definition of while

Assume A, B : Set.

$$\frac{b : B \qquad\qquad p : B \to (\mathrm{IO}(A) + \mathrm{IO}^{+}(B))}{\mathsf{while}_B(b, p) : \mathrm{IO}(A)}$$

- If $p(b) = \mathsf{i}(q)$ then
  $\mathsf{while}(b, p) = q$

- If $q(b) = \mathsf{j}(q)$ then
  $\mathsf{while}(b, p) = q^{-} * \lambda b'.\mathsf{while}(b', p)$

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$(c, p), q) = q(c)^{-} * \lambda r.$redirect$(p(r), q)$.

2.3. Equality

Equality corresponding to extensional equality on non-well-founded trees:
Bisimulation (definition according I. Lindström):

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

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

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

$\mathrm{Eq}'(n, \mathrm{leaf}(a), \mathrm{do}(c,p))$
$= \mathrm{Eq}'(n, \mathrm{do}(c,p), \mathrm{leaf}(a)) = \bot$

$\mathrm{Eq}'(n, \mathrm{leaf}(a), \mathrm{leaf}(a')) = \mathrm{I}(A, a, a')$.

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

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

- Eq is the natural extension of extensional equality to non-well-founded trees (if we take for I 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: No normalization

Let $A = B = C =$ N, R(c) arbitrary.
Assume f : N $\rightarrow$ N.
$p := \lambda n.\mathsf{do}(f(n), \lambda x.\mathsf{leaf}(n + 1))$: N $\rightarrow$ IO(B)
$q := \lambda p.\mathsf{j}(p^{+})$: N $\rightarrow$ (IO(A) + IO^{+}(B)).

$$
\begin{aligned}
&\mathsf{while}(0, q) \\
\longrightarrow\ \ & \mathsf{while}'(p(0), q) \\
\longrightarrow\ \ & \mathsf{do}(f(0), \lambda x.\mathsf{while}'(\mathsf{leaf}(1), q)) \\
\longrightarrow\ \ & \mathsf{do}(f(0), \lambda x.\mathsf{while}'(p(1), q)) \\
\longrightarrow\ \ & \mathsf{do}(f(0), \lambda x.\mathsf{do}(f(1), \lambda y.\mathsf{while}'(\mathsf{leaf}(2), q))) \\
\longrightarrow\ \ & \cdots \\
\longrightarrow\ \ & \mathsf{do}(f(0), \lambda x.\mathsf{do}(f(1), \lambda y.\mathsf{do}(f(2), \lambda z. \cdots)))
\end{aligned}
$$

Consequence: with intensional equality type-checking undecidable.

3. Normalizing version

Add while as a constructor.

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

Restrict A, B to be elements of a universe.
(Restriction of B would suffice).

Assume
U : Set, T : U → Set.
Assume $w = (C, R)$ is a world.

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

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

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

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

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

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

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

(The rule with occurrences of $(+)$ denotes two rules:
One where everywhere $(+)$ is replaced by $+$ and one where $(+)$ is omitted).

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

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

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

- $\text{emb}(\text{leaf}(a)) = \text{leaf}(a)$.
- $\text{emb}^{(+)}(\text{do}^{(+)}(c, p)) = \text{do}^{(+)}(c, \lambda x.\text{emb}(p(x)))$.
- $\text{emb}(\text{while}_{\hat{B}}(b, p)) =$
  $\text{while}_B(b, \lambda x.\text{emb}'(p(x)))$
  with $\text{emb}'(\text{i}(p)) = \text{i}(\text{emb}(p))$,
  $\text{emb}'(\text{j}(p)) = \text{j}(\text{emb}^{+}(p))$.

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

Decompose:

Define decompose : $\mathrm{IO}_{\mathsf{wf}}(A) \to$
$$A + \Sigma c : C.(R(c) \to \mathrm{IO}_{\mathsf{wf}}(A))$$
s.t.

If emb(p) = leaf(a),
    decompose(p) = i(a).
If emb(p) = do(c, q),
    then decompose(p) = j(c, q') where q' s.t.
    emb($q'(x)$) = $q(x)$.

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 depend on the state of knowledge.

States = "objective knowledge" about the devices.

The state is influenced by commands, e.g.
- open a new window.
- switch on a printer.
- test whether the printer is switched on.

A world is now a quadruple (S, C, R, ns) s.t.
- S : Set (set of states).
- $C : S \rightarrow$ Set (set of commands).
- $R : (s : S, C(s)) \rightarrow$ 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.

$$\frac{A : S \to \mathsf{Set} \qquad\qquad s : S}{\mathsf{IO}(A, s) : \mathsf{Set}}$$

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

$$\frac{s : S \qquad\qquad a : A(s)}{\mathsf{leaf}(a) : \mathsf{IO}(A, s)}$$

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

$$\frac{s : S \qquad\qquad a : A(s)}{\widetilde{\eta}_a^A(s) : \mathsf{IO}(A, s)}$$

$\widetilde{\eta}_a^A(s) = \mathsf{leaf}(a)$.

$$\frac{\begin{array}{c} s : S \\ p : \mathsf{IO}(A, s) \\ B : S \to \mathsf{Set} \\ q : (s : S, a : A(s)) \to \mathsf{IO}(B, s) \end{array}}{p \mathbin{\widetilde{*}_s^{A,B}} q : \mathsf{IO}(B, s)}$$

$\mathsf{do}(c, p) \mathbin{\widetilde{*}} q = \mathsf{do}(c, \lambda r.(p(r) \mathbin{\widetilde{*}} q))$.
$\mathsf{leaf}(a) \mathbin{\widetilde{*}} q = q(s, a)$.

Corresponding monad

Consider $\mathrm{IO} : (S \to \mathsf{Set}) \to (S \to \mathsf{Set})$.

$\eta : (A : S \to \mathsf{Set}, a : (s : S) \to A(s)) \to \mathrm{IO}(A)$,
$\eta^A_a := \lambda s.\widetilde{\eta}^A_{a(s)}(s)$.

$\mu : (A : S \to \mathsf{Set}, p : \mathrm{IO}(\mathrm{IO}(A)) \to \mathrm{IO}(A)$,
$\mu^A(p) := \lambda s.(p(s) \mathbin{\widetilde{*}}^{\mathrm{IO}(A),A}_s (\lambda s, q.q))$.

$\mathsf{map} : (A, B : S \to \mathsf{Set},$
$\quad f : (s : S, a : A(s)) \to B(s),$
$\quad p : \mathrm{IO}(A))$
$\quad \to \mathrm{IO}(B),$
$\mathsf{map}^{A,B}(f, p) := \lambda s.(p(s) \widetilde{*}^{A,B}_s \lambda s, a.\mathsf{leaf}(f(s, a)))$.

This yields a monad on presheaves over the discrete category S.

Corresponding $*$-operation:

$$\begin{aligned}
* : (&A, B : S \to \mathsf{Set}, \\
&\quad p : \mathrm{IO}(A), \\
&\quad q : (s : S, A(s)) \to \mathrm{IO}(B(s))) \\
&\to \mathrm{IO}(B), \\
(p *^{A,B} q)(s) &= p(s) \,\tilde{*}_s^{A,B}\, q.
\end{aligned}$$

While

$\mathrm{IO}^+(A, s)$ defined as before.

$$\frac{\begin{array}{c} B : S \to \mathsf{Set} \\ s : S \\ b : B(s) \\ q : (s : S, b : B(s)) \to (\mathrm{IO}(A, s) + \mathrm{IO}^+(B, s)) \end{array}}{\mathsf{while}_{B,s}(b, q) : \mathrm{IO}(A, s)}$$

If $q(s, b) = \mathsf{i}(p)$ then
$\mathsf{while}_{B,s}(b, q) = p$.

If $q(s, b) = \mathsf{j}(p)$ then
$\mathsf{while}_{B,s}(b, q) = p^- * \lambda s', b'.\mathsf{while}_{B,s'}(b', q)$.

Redirect

Assume

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

Define

$\mathrm{redirect}_{w,w'}(A, Rel, q, s, s', rel, p)$
  $: \mathrm{IO}_{w'}(\lambda s''.\Sigma s : S.(Rel(s, s'') \wedge A(s)))$

by

$$\mathsf{redirect}_{w,w'}(A, Rel, q, s, s', rel, \mathsf{leaf}(a)) = \\ \quad \mathsf{leaf}(<s, rel, a>).$$

$$\mathsf{redirect}_{w,w'}(A, Rel, q, s, s', rel, \mathsf{do}(c, p)) = \\ \quad q(s, c, s', rel)^{-} * \\ \quad \lambda s'', <r, rel'>. \\ \quad \quad \mathsf{redirect}_{w,w'}(A, Rel, q, ns(s, c, r), s'', rel', p(r)).$$

execute

Let $w_0 = (S_0, C_0, R_0, ns_0)$ be a standard world, $s_0 : S$ be a state which corresponds to the existence of knowledge about the environment. Assume $p : \mathrm{IO}_{w_0}(A, s_0)$.

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

(If one has a program which requires a certain state s of the environment, compose before it a program, which starts from the initial state, and making tests of the environment tries to move to state s; if it fails it terminates. Execute the result).

5. Parallelism, Non-determinism

Non-determinism

Additional constructor of IO, of the same form as do.

$R(s, c)$ is now the answer of the oracle, which does non-deterministic choice.

Modify $\mathrm{IO}^+(A, s)$ s.t. every execution has at least one "real" command.

Parallelism

Add to our world:

A set of parallel commands

$$NC : S \to \mathrm{Set},$$

an index set of processes for every command

$$ND : (s : S, c : NC(s)) \to \mathrm{Set},$$

a world for every process

$$Nw : (s : S, c : NC(s), d : ND(s, c)) \\ \to world$$

a result type of each process

$$NR : (s : S, c : NC(s), d : ND(s, c), \\ s' : Nw(s, c, d).S) \\ \to \mathrm{Set},$$

a next state depending on the final states of all processes,

$$Nns : (s : S, \\ c : NC(s), \\ s' : (d : ND(s, c)) \to NW(s, c, d).S, \\ r : (d : ND(s, c)) \to NR(s, c, d, s'(d))) \\ \to S.$$

Processes can communicate via commands in their worlds.

New constructor

$$
\begin{aligned}
\text{parallel} : (&s : S,\\
&c : NC(s),\\
&p : (d : ND(s,c)) \to \mathrm{IO}_{w(s,c,d)}(NR(s,c,d)),\\
&np : (s' : (d : ND(s,c)) \to w(s,c,d).S,\\
&\qquad r : (d : ND(s,c)) \to NR(s,c,d,s'(d)))\\
&\qquad \to \mathrm{IO}_w(A, ns(s,c,s',r)))\\
&\to \mathrm{IO}_w(A,s).
\end{aligned}
$$

Further construction: Parallelism with dependency only on the first process which stops.

(Then Nns will have type:

$$
\begin{aligned}
Nns : (&s : S, c : NC(s), d : ND(s,c),\\
&s' : NW(s,c,d).S, r : NR(s,c,d,s'))\\
&\to S).
\end{aligned}
$$

and parallel is defined accordingly).

Conclusion

- Inductive definition of the IO-monad by IO-trees.
- Parameterized over worlds (over input/output).
- New constructions: run, redirect.
- Extensions to state-dependent command sets.
- Nondeterminism, parallelism.