

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

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

with abbreviations

$$\eta_a := \eta_a^A := \eta(A, a),$$

$$p * q := p *_{A,B} q := *(A, B, p, q),$$

s.t. for $A, B, C : \text{Set}, a : A, p : M(A),$
 $q : A \rightarrow M(B), r : B \rightarrow M(C)$:

- $\eta_a * q = q(a)$.
- $p * \lambda x. \eta_x = p$.
- $(p * q) * r = p * \lambda x. (q(x) * r)$.

IO-Monad

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

- $\text{IO}(A)$ = 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

$\text{input}(d, A) : \text{IO}(A)$

input from device d an element $a : A$
and return a .

$\text{output}(d, A) : A \rightarrow \text{IO}(1)$

for $a : A$ output a on device d
and return $\langle \rangle : 1$.

IO-Monad in Haskell:

Small part of the program interactive.

Large part purely functional.

Problems of the IO-Monad:

- * cannot be a constructor.
 - ⇒ 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).
- $R : C \rightarrow \text{Set}$ (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-well-founded) trees with

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

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

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

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

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

New 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 $p : \text{IO}_{w_0}(A)$ 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 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:

$$\text{interact} : (c : C) \rightarrow \text{IO}(R(c)).$$

$$\text{interact}(c) = \text{do}(c, \lambda x.\text{leaf}(x)).$$

$\text{interact}(c)$ 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 $b : B$
- $q : B \rightarrow (\text{IO}(A) + \text{IO}(B))$.

$\text{while}_B(b, q) : \text{IO}(A)$ does the following:

- If $q(b)$ is in $\text{IO}(B)$ then it runs this program. If it terminates with leaf b' , it continues with $\text{while}_B(b', q)$.
- If $q(b)$ is in $\text{IO}(A)$ 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:

$$\frac{A : \text{Set}}{\text{IO}^+(A) : \text{Set}} \quad \frac{a : \text{IO}^+(A)}{a^- : \text{IO}(A)}$$

$$\frac{c : C \quad p : R(c) \rightarrow \text{IO}(A)}{\text{do}^+(c, p) : \text{IO}^+(A)}$$

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

Definition of while

Assume $A, B : \text{Set}$.

$$\frac{b : B \quad p : B \rightarrow (\text{IO}(A) + \text{IO}^+(B))}{\text{while}_B(b, p) : \text{IO}(A)}$$

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

2.2. Redirect

Assume

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

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

$\text{redirect}(\text{leaf}(a), q) = \text{leaf}(a)$.

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

2.3. Equality

Equality corresponding to extensional equality on non-well-founded trees:

Bisimulation (definition according I. Lindström):

$$\frac{p : \text{IO}(A) \quad q : \text{IO}(A)}{\text{Eq}(p, q) : \text{Set}}$$

$$\frac{p : \text{IO}(A) \quad q : \text{IO}(A) \quad n : \mathbb{N}}{\text{Eq}'(n, p, q) : \text{Set}}$$

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

$$\begin{aligned} & \text{Eq}'(n, \text{leaf}(a), \text{do}(c, p)) \\ &= \text{Eq}'(n, \text{do}(c, p), \text{leaf}(a)) = \perp \end{aligned}$$

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

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

$$\begin{aligned} & \text{Eq}'(\text{S}(n), \text{do}(c, p), \text{do}(c', p')) = \\ & \quad \Sigma q : \text{I}(C, c, c'). \forall r : R(c). \text{Eq}(n, p(r), p'(\dots r \dots)). \end{aligned}$$

- 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.
 - ⇒ Extensionally, for every IO-behaviour there is exactly one program.
 - ⇒ IO-tree = suitable model of IO.

Problem: No normalization

Let $A = B = C = \mathbb{N}$, $R(c)$ arbitrary.

Assume $f : \mathbb{N} \rightarrow \mathbb{N}$.

$p := \lambda n. \text{do}(f(n), \lambda x. \text{leaf}(n + 1)) : \mathbb{N} \rightarrow \text{IO}(B)$

$q := \lambda p. \text{j}(p^+) : \mathbb{N} \rightarrow (\text{IO}(A) + \text{IO}^+(B)).$

$\text{while}(0, q)$

$\longrightarrow \text{while}'(p(0), q)$

$\longrightarrow \text{do}(f(0), \lambda x. \text{while}'(\text{leaf}(1), q))$

$\longrightarrow \text{do}(f(0), \lambda x. \text{while}'(p(1), q))$

$\longrightarrow \text{do}(f(0), \lambda x. \text{do}(f(1), \lambda y. \text{while}'(\text{leaf}(2), q)))$

$\longrightarrow \dots$

$\longrightarrow \text{do}(f(0), \lambda x. \text{do}(f(1), \lambda y. \text{do}(f(2), \lambda z. \dots)))$

Consequence: with intensional equality type-checking undecidable.

3. Normalizing version

Add while as a constructor.

Problem: while refers to $\text{IO}(A) + \text{IO}^+(B)$.
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

$U : \text{Set}, T : U \rightarrow \text{Set}$.

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

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

$$\frac{\hat{A} : \mathbf{U}}{\text{IO}_w(\hat{A}) : \text{Set}}$$

$$\frac{\hat{A} : \mathbf{U}}{\text{IO}_w^+(\hat{A}) : \text{Set}}$$

$$\frac{p : \text{IO}^+(\hat{A})}{p^- : \text{IO}(\hat{A})}$$

$$\frac{a : A}{\text{leaf}(a) : \text{IO}(\hat{A})}$$

$$\frac{c : C \quad p : R(c) \rightarrow \text{IO}(\hat{A})}{\text{do}^{(+)}(c, p) : \text{IO}^{(+)}(\hat{A})}$$

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

$$\frac{\hat{B} : \mathbf{U} \quad b : B \quad p : B \rightarrow (\text{IO}(\hat{A}) + \text{IO}^+(\hat{B}))}{\text{while}_{\hat{B}}(b, p) : \text{IO}(\hat{A})}$$

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

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

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

Decompose:

Define $\text{decompose} : \text{IO}_{\text{wf}}(A) \rightarrow$
 $A + \Sigma c : C.(R(c) \rightarrow \text{IO}_{\text{wf}}(A))$

s.t.

If $\text{emb}(p) = \text{leaf}(a)$,
 $\text{decompose}(p) = i(a)$.

If $\text{emb}(p) = \text{do}(c, q)$,
 then $\text{decompose}(p) = j(c, q')$ where q' s.t.
 $\text{emb}(q'(x)) = q(x)$.

Execute(p) does now the following:

- If $\text{decompose}(p) = i(a)$, then terminate with result a .
- If $\text{decompose}(p) = j(\langle c, q \rangle)$, 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).
- $C : S \rightarrow \text{Set}$ (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.

$$\frac{A : S \rightarrow \text{Set} \quad s : S}{\text{IO}(A, s) : \text{Set}}$$

Assume $A : S \rightarrow \text{Set}$.

$$\frac{s : S \quad a : A(s)}{\text{leaf}(a) : \text{IO}(A, s)}$$

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

$$\frac{s : S \quad a : A(s)}{\tilde{\eta}_a^A(s) : \text{IO}(A, s)}$$

$$\tilde{\eta}_a^A(s) = \text{leaf}(a).$$

$$\frac{\begin{array}{l} s : S \\ p : \text{IO}(A, s) \\ B : S \rightarrow \text{Set} \\ q : (s : S, a : A(s)) \rightarrow \text{IO}(B, s) \end{array}}{p \tilde{*}_s^{A, B} q : \text{IO}(B, s)}$$

$$\text{do}(c, p) \tilde{*} q = \text{do}(c, \lambda r. (p(r) \tilde{*} q)).$$

$$\text{leaf}(a) \tilde{*} q = q(s, a).$$

Corresponding monad

Consider $\text{IO} : (S \rightarrow \text{Set}) \rightarrow (S \rightarrow \text{Set})$.

$$\eta : (A : S \rightarrow \text{Set}, a : (s : S) \rightarrow A(s)) \rightarrow \text{IO}(A),$$

$$\eta_a^A := \lambda s. \tilde{\eta}_{a(s)}^A(s).$$

$$\mu : (A : S \rightarrow \text{Set}, p : \text{IO}(\text{IO}(A)) \rightarrow \text{IO}(A),$$

$$\mu^A(p) := \lambda s. (p(s) \tilde{*}_s^{\text{IO}(A), A} (\lambda s, q. q)).$$

$$\text{map} : (A, B : S \rightarrow \text{Set},$$

$$f : (s : S, a : A(s)) \rightarrow B(s),$$

$$p : \text{IO}(A))$$

$$\rightarrow \text{IO}(B),$$

$$\text{map}^{A, B}(f, p) := \lambda s. (p(s) \tilde{*}_s^{A, B} \lambda s, a. \text{leaf}(f(s, a))).$$

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

Corresponding $*$ -operation:

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

While

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

$$\frac{\begin{array}{l} B : S \rightarrow \text{Set} \\ s : S \\ b : B(s) \\ q : (s : S, b : B(s)) \rightarrow (\text{IO}(A, s) + \text{IO}^+(B, s)) \end{array}}{\text{while}_{B,s}(b, q) : \text{IO}(A, s)}$$

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

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

Redirect

Assume

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

Define

$$\begin{aligned} & \text{redirect}_{w, w'}(A, Rel, q, s, s', rel, p) \\ & : \text{IO}_{w'}(\lambda s'' . \sum s : S. (Rel(s, s'') \wedge A(s))) \end{aligned}$$

by

$$\text{redirect}_{w,w'}(A, Rel, q, s, s', rel, \text{leaf}(a)) = \text{leaf}(\langle s, rel, a \rangle).$$

$$\begin{aligned} \text{redirect}_{w,w'}(A, Rel, q, s, s', rel, \text{do}(c, p)) = \\ q(s, c, s', rel)^- * \\ \lambda s'', \langle r, rel' \rangle. \\ \text{redirect}_{w,w'}(A, Rel, q, ns(s, c, r), s'', rel', p(r)). \end{aligned}$$

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 $p : IO_{w_0}(A, s_0)$.

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 $\text{IO}^+(A, s)$ s.t. every execution has at least one “real” command.

Parallelism

Add to our world:

A set of parallel commands

$$NC : S \rightarrow \text{Set},$$

an index set of processes for every command

$$ND : (s : S, c : NC(s)) \rightarrow \text{Set},$$

a world for every process

$$Nw : (s : S, c : NC(s), d : ND(s, c)) \\ \rightarrow \text{world}$$

a result type of each process

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

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

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

Processes can communicate via commands in their worlds.

New constructor

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

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

(Then Nns will have type:

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

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.