Interactive Programs in Dependent Type Theory

Anton Setzer, Uppsala

(Joint work with Peter Hancock, Edinburgh)

Sept. 18, 1999

1. IO-trees.
2. Constructions for defining IO-trees.

(3. Normalizing version.

4. State-dependent IO.)

1. IO-trees

Problem: Ordinary programs in type theory are functions.
- One input.
- One output.

Goal: Addition of Interactive Programs
Models for Input/Output:

1) Streams.

Inputstream = I × Inputstream.

Largest fixed point.

Elements: $<i_0, <i_1, <i_2, , \ldots >>>$

Outputstream = O × Outputstream.

Largest fixed point.

Elements: $<o_0, <o_1, <o_2, , \ldots >>>$

Interactive programs =

Inputstream → Outputstream.

Problem:

- Additional concept of coinductive definitions necessary.
- Difficulties with unbounded many input/output devices
- Timing between input/output depends on evaluation strategy.

2) The IO-Monad

The IO-monad is a triple $(\mathrm{IO}, \eta, *)$, s.t.:

- IO : Set $\rightarrow$ Set.

  $\mathrm{IO}(A) =$ set of interactive programs, which, if they terminate, return an element $a : A$.

- $\eta : (A : \mathrm{Set}, a : A) \rightarrow \mathrm{IO}(A)$.

  η_a^A: no interaction, returns a.

- $* : (A : \mathrm{Set}, B : \mathrm{Set}, p : \mathrm{IO}(A), q : A \rightarrow \mathrm{IO}(B))$
  $\rightarrow \mathrm{IO}(B)$.

  $p *_{A,B} q$ starts with p.
  If p returns a, then it continues with $q(a)$ and returns its result.

Abbreviations

- $\eta_a := \eta_a^A$,
- $p * q := p *_{A,B} q$.

Laws

Let A, B, C : Set, $a : A, p :$ IO(A),
$q : A \to$ IO$(B), r : B \to$ IO(C):

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

To get real programs, add constructions like

input(d) : $\mathrm{IO}(\mathrm{I}_d)$

    input from input-device d an element $a : \mathrm{I}_d$
    and return a.

output(d) : $\mathrm{O}_d \to \mathrm{IO}(\{*\})$

    for $a : \mathrm{O}_d$ output a on output-device d
    and return $*$ (= success).

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.

3) Our Definition of IO-programs: The IO-tree

Worlds

A world w is a pair (C, R) s.t.

- C : Set (Commands).
- $R : C \to$ Set (responses to a command).

Example:

```
C = data { readstr, writestr(s: string)}
  : Set

R: C -> Set,
R(readstring)      = string
R(writestring(s)) = {*}
```

IO-trees

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

Execution of IO-programs:

Add operation execute.

Status:
- Like function "compute head normal 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.

Example: “Hello world”

```
C = data { readstr, writestr(s: string)}
  : Set

R: C -> Set
R(readstring)      = string
R(writestring(s)) = {*}

helloworld
= do readstring
     \s.if (s = "Hello")
        then (do
              (writestring "World")
              \a.leaf success)
        else (leaf fail)
: IO({success,fail})
```

2. Constructions for Defining IO-trees

2. 1. Definition of η, $*$

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

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

2.2. While

Assume:

- Sets A, B,
- an initial value $a : A$
- $p : A \to (\mathrm{IO}(A) + \mathrm{IO}(B))$.

$\text{while}_{A,B}(a, p) : \mathrm{IO}(B)$ does the following:

- If $p(a)$ is in $\mathrm{IO}(A)$ then it carries out this program.
  If it terminates with result a', it continues with $\text{while}_B(a', p)$.
- If $p(a)$ is in $\mathrm{IO}(B)$ then it carries out 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:

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

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

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

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

Definition of while

Assume A, B : Set.

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

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

2.3. Repeat

Assume:

- Sets A, B,
- an initial value $a : A$
- $p : A \to (\mathrm{IO}^{+}(A + B))$.

repeat$_{A,B}(a, p) : \mathrm{IO}(B)$ does the following:

- It carries out $p(a)$.

  If the result is $a' : A$ it repeats the loop starting with a'.

  If the result is $b : B$, it terminates with b.

Assume A, B : Set.

$$\frac{a : A \qquad\qquad p : A \to \mathrm{IO}^{+}(A + B)}{\mathsf{repeat}_{A,B}(a, p) : \mathrm{IO}(B)}$$

$$\begin{aligned}\mathsf{repeat}(a, p) = p(a)^{-} * \lambda c.\mathsf{case}\ c\ \mathsf{of}\ &\{\mathsf{inl}(a') \to \mathsf{repeat}(a', p),\\ &\ \mathsf{inr}(b) \to \mathsf{leaf}(b)\}.\end{aligned}$$

Exercise: Reduce repeat to while.

Example: A rudimentary editor.

```
C = data{ readchar} : Set

R : C -> Set
R(c) = data{ch(c: char), cursorleft,
            terminate}

editor
= repeat
  C R string string ""
  (\s -> do
        readchar
        \l -> case l of {
                ch c
                -> leaf (inl (cons c s)),
                cursorleft
                -> leaf (inl (truncate s)),
                terminate
                -> leaf (inr s)}
```

2.4. 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$(p(r), q)$.

Example

Highlevel world w_0:

```
C0 = data{ readstring, writestring(s: string)}
   : Set

R0 : C0 -> Set
R0(readstring)  = string
R0(writestring) = {*}
```

Lowlevel world w_1:

```
C1 = data{readkey, writesymbol(l: char),
          movecursorleft, movecursorright}

R1: C1 -> Set
R1(readkey) = char
              + {cursorleft, cursorright, Escape}
R1(writesymbol l)   = {*}
R1(movecursorleft)  = {*}
R1(movecursorright) = {*}
```

Redirect programs in w_0 to programs in w_1 by

```
q: (c: C0) -> IO+ w1 (R0 c)
q(readstring)     = some editor
                  : IO+ w1 string
q(writestring s) = some outputroutine for s
                  : IO+ w1 {*}
```

(optional)

2.5. Equality

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

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

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

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

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

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

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

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

- Eq 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 $A = B = C = \mathsf{N}$, $\mathsf{R}(c)$ arbitrary.

Assume $f : \mathsf{N} \to \mathsf{N}$.
$p := \lambda n.\mathsf{inl}(\mathsf{do}^{+}(f(n), \lambda y.\mathsf{leaf}(n+1)))$
$\quad : \mathsf{N} \to (\mathrm{IO}^{+}(A) + \mathrm{IO}(B))$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\mathsf{while}(0, p)} \\
\longrightarrow & \mathsf{do}(f(0), \lambda x.\mathsf{leaf}(1)) * \lambda m.\mathsf{while}(m, p) \\
\longrightarrow & \mathsf{do}(f(0), \lambda x.(\mathsf{leaf}(1) * \lambda m.\mathsf{while}(m, p))) \\
\longrightarrow & \mathsf{do}(f(0), \lambda x.(\mathsf{while}(1, p))) \\
\longrightarrow & \mathsf{do}(f(0), \lambda x.(\mathsf{do}(f(1), \lambda x.\mathsf{while}(2, p)))) \\
\longrightarrow & \mathsf{do}(f(0), \lambda x.(\mathsf{do}(f(1), \lambda x.(\mathsf{do}(f(2), \\
 & \qquad \lambda x.\mathsf{while}(3, p)))))) \\
\longrightarrow & \cdots
\end{array}
$$

Consequence: with intensional equality type-checking undecidable.

3. Normalizing version

Add while as a constructor.

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

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

For simplicity not in this lecture.

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

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

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

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

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

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

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

- $\mathrm{emb}(\mathrm{leaf}(a)) = \mathrm{leaf}(a)$.
- $\mathrm{emb}^{(+)}(\mathrm{do}^{(+)}(c, p)) = \mathrm{do}^{(+)}(c, \lambda x.\mathrm{emb}(p(x)))$.
- $\mathrm{emb}(\mathrm{while}_B(b, p)) =$
  $\mathrm{while}_B(b, \lambda x.\mathrm{emb}'(p(x)))$
  with $\mathrm{emb}'(\mathrm{inl}(p)) = \mathrm{inl}(\mathrm{emb}(p))$,
  $\mathrm{emb}'(\mathrm{inr}(p)) = \mathrm{inr}(\mathrm{emb}^{+}(p))$.

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

Decompose:

Define

$\text{decompose} : \text{IO}_{\text{wf}}(A)$

$\quad\quad \to A + \Sigma c : C.(R(c) \to \text{IO}_{\text{wf}}(A))$

s.t.:

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

  $\text{decompose}(p) = \text{inl}(a)$.

If $\text{emb}(p) = \text{do}(c, q)$,

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

Execute(p) does the following:

- If $\text{decompose}(p) = \text{inl}(a)$, then terminate with result a.
- If $\text{decompose}(p) = \text{inr}(<c, q>)$, 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.

Worlds with State-dependency

A world is a quadruple (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.

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

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

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

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

Composition of Programs

Let $A, B : S \to \mathsf{Set}$,

$$\frac{\begin{array}{c} s_0 : S \\ p : \mathrm{IO}(A, s) \\ q : (s : S, a : A(s)) \to \mathrm{IO}(B, s) \end{array}}{p *_{s_0}^{A,B} q : \mathrm{IO}(B, s)}$$

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

While

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

$$\frac{\begin{array}{c} B : S \to \mathsf{Set} \\ s_0 : S \\ b : B(s_0) \\ q : (s : S, b : B(s)) \to (\mathrm{IO}^+(B, s) + \mathrm{IO}(A, s)) \end{array}}{\mathsf{while}_{B,s_0}(b, q) : \mathrm{IO}(A, s)}$$

If $q(s_0, b) = \mathsf{inl}(p)$ then
$\quad \mathsf{while}_{B,s_0}(b, q) = p^- * \lambda s', b'.\mathsf{while}_{B,s'}(b', q)$.

If $q(s_0, b) = \mathsf{inr}(p)$ then
$\quad \mathsf{while}_{B,s_0}(b, q) = p$.

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).

Conclusion

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

Possible Extensions:

- Nondeterminism,
- parallelism.