What I'm doing

and why

Peter Hancock

Jan 2007

It's all about IO

- I live in type-theory (a la Martin-Löf):

proofs	programs
propositions	specifications

It's all about IO

- I live in type-theory (a la Martin-Löf):

proofs	programs
propositions	specifications

- I've been a programmer, in control systems, machine architecture, transaction processing, file systems.

It's all about IO

- I live in type-theory (a la Martin-Löf):

proofs	programs
propositions	specifications

- I've been a programmer, in control systems, machine architecture, transaction processing, file systems.
- If programs are proofs, and I write a program that controls a machine, what theorem is it I have proved?

It's all about IO

- I live in type-theory (a la Martin-Löf):

proofs	programs
propositions	specifications

- I've been a programmer, in control systems, machine architecture, transaction processing, file systems.
- If programs are proofs, and I write a program that controls a machine, what theorem is it I have proved?
- What does it mean to *run* a program? (I was trained in philosophy).

It's all about IO

- I live in type-theory (a la Martin-Löf):

proofs	programs
propositions	specifications

- I've been a programmer, in control systems, machine architecture, transaction processing, file systems.
- If programs are proofs, and I write a program that controls a machine, what theorem is it I have proved?
- What does it mean to *run* a program? (I was trained in philosophy).
- 'Running' $\neq$ evaluating. (Fetch/Execute.)

Interaction structures

Handshaken (command response) interfaces.

States	$S : \mathrm{Set}$
Commands	$C(s) : \mathrm{Set}\ (s \in S)$
Responses	$R(s, c) : \mathrm{Set}\ (s \in S, c \in C(s))$
next state	$n(s, c, r) \in S\ (s \in S, c \in C(s), r \in R(s, c))$

Interaction structures

Handshaken (command response) interfaces.

States	S : Set
Commands	$C(s)$: Set $(s \in S)$
Responses	$R(s, c)$: Set $(s \in S, c \in C(s))$
next state	$n(s, c, r) \in S$ $(s \in S, c \in C(s), r \in R(s, c))$

Gives a *predicate transformer*

$$P \subseteq S \mapsto \{\, s \in S \mid (\sum c \in C(s))\,(\prod r \in R(s, c))\, P(n(s, c, r) \,\}$$

That's reassuring! (Dijkstra, Hoare, refinement calculus, ...)

Interaction structures

Handshaken (command response) interfaces.

States	S : Set
Commands	$C(s)$: Set $(s \in S)$
Responses	$R(s,c)$: Set $(s \in S, c \in C(s))$
next state	$n(s,c,r) \in S$ $(s \in S, c \in C(s), r \in R(s,c))$

Gives a *predicate transformer*

$$P \subseteq S \mapsto \{\, s \in S \mid (\sum c \in C(s))\,(\prod r \in R(s,c))\, P(n(s,c,r)\,\}$$

That's reassuring! (Dijkstra, Hoare, refinement calculus, ...)

- These are essentially indexed containers.
- Another thing they are is *coverings* – in topology.

Some work with Pierre Hyvernat on 'pre-topology' and linear logic (*re* simulations); also computational meaning of locale conditions.

Eating: Benedictus, benedicat

An old, venerable model of IO in functional programming is stream processing.

Eating: Benedictus, benedicat

An old, venerable model of IO in functional programming is stream processing. I had, at the back of my head, a way of representing continuous functions on streams (coming from intuitionism: ‘The Bar Theorem’ (Brouwer)).

Eating: Benedictus, benedicat

An old, venerable model of IO in functional programming is stream processing. I had, at the back of my head, a way of representing continuous functions on streams (coming from intuitionism: 'The Bar Theorem' (Brouwer)). I mentioned it to Neil in the pub:

$$
\begin{array}{ll}
A^\omega \to_c B & T_A(B) \triangleq (\mu\, X)\, B + X^A \\
A^\omega \to_c B^\omega & P_A(B) \triangleq (\nu\, X)\, T_A(B \times X) \\
(\cdot) : \ldots & \otimes : P_B C \times P_A B \to P_A C
\end{array}
$$

Eating: Benedictus, benedicat

An old, venerable model of IO in functional programming is stream processing. I had, at the back of my head, a way of representing continuous functions on streams (coming from intuitionism: 'The Bar Theorem' (Brouwer)). I mentioned it to Neil in the pub:

$$
\begin{array}{ll}
A^\omega \rightarrow_c B & T_A(B) \triangleq (\mu\, X)\, B + X^A \\
A^\omega \rightarrow_c B^\omega & P_A(B) \triangleq (\nu\, X)\, T_A(B \times X) \\
(\cdot) : \ldots & \otimes : P_B C \times P_A B \rightarrow P_A C
\end{array}
$$

Some overlap with what other people are doing, who express some interest.

Eating: Benedictus, benedicat

An old, venerable model of IO in functional programming is stream processing. I had, at the back of my head, a way of representing continuous functions on streams (coming from intuitionism: 'The Bar Theorem' (Brouwer)). I mentioned it to Neil in the pub:

$$
\begin{array}{ll}
A^\omega \rightarrow_c B & T_A(B) \triangleq (\mu X)\, B + X^A \\
A^\omega \rightarrow_c B^\omega & P_A(B) \triangleq (\nu X)\, T_A(B \times X) \\
(\cdot) : \ldots & \otimes : P_B C \times P_A B \rightarrow P_A C
\end{array}
$$

Some overlap with what other people are doing, who express some interest. But there's more!

Eating: Benedicto, benedicatur

Final coalgebras, by their very nature often (but not always. . .) have a nice stream-like topology. In particular this goes for containers:

$$(S \lhd P)\, X \triangleq (\sum s \in S)\, X^{P(s)}$$

Eating: Benedicto, benedicatur

Final coalgebras, by their very nature often (but not always. . .) have a nice stream-like topology. In particular this goes for containers:

$$(S \lhd P)\, X \triangleq (\sum s \in S)\, X^{P(s)}$$

There's a representation of continuous function $\nu(F) \to_c \nu(G)$ of the same general form as the stream case:

$$(\nu \,\ldots)\, (\mu \,\ldots)\, ((\sum \,\ldots)\, \ldots) + ((\prod \,\ldots)\, \ldots)$$

Eating: Benedicto, benedicatur

Final coalgebras, by their very nature often (but not always. . .) have a nice stream-like topology. In particular this goes for containers:

$$(S \lhd P)\,X \triangleq (\sum s \in S)\,X^{P(s)}$$

There's a representation of continuous function $\nu(F) \to_c \nu(G)$ of the same general form as the stream case:

$$(\nu\ \ldots)\,(\mu\ \ldots)\,((\sum\ \ldots)\ \ldots) + ((\prod\ \ldots)\ \ldots)$$

In working this out, one uses (dependent types and) *induction-recursion* in an essential way.

Eating: Benedicto, benedicatur

Final coalgebras, by their very nature often (but not always. . .) have a nice stream-like topology. In particular this goes for containers:

$$(S \lhd P)\,X \triangleq (\textstyle\sum s \in S)\,X^{P(s)}$$

There's a representation of continuous function $\nu(F) \to_c \nu(G)$ of the same general form as the stream case:

$$(\nu\ \ldots)\,(\mu\ \ldots)\,((\textstyle\sum\ \ldots)\ \ldots) + ((\textstyle\prod\ \ldots)\ \ldots)$$

In working this out, one uses (dependent types and) *induction-recursion* in an essential way.
To me, this is a case study for the kind of coinduction one needs in dependent type theory. (Which is a topic that needs exploration.)

Turing machines and coalgebras?

I *might* do something on this. (For much the same reasons: Neil likes it . . .)

Turing machines and coalgebras?

I *might* do something on this. (For much the same reasons: Neil likes it . . .)
But I might work on some more intricate, meta-mathematical things. It is important (for programming) to get the theory, in place, and maybe experiment in a non-standard direction . . . .

Turing machines and coalgebras?

I *might* do something on this. (For much the same reasons: Neil likes it . . .)

But I might work on some more intricate, meta-mathematical things. It is important (for programming) to get the theory, in place, and maybe experiment in a non-standard direction . . . .

But still, what is it to 'run' a program??