Fast and Loose Reasoning is Morally Correct

Nils Anders Danielsson Jeremy Gibbons
John Hughes Patrik Jansson

Chalmers and Oxford

Sloppy proofs

- Many proofs assume language is total.
  - No non-termination.
  - No bottoms.
  - Often also no infinite values.

Sloppy proofs

- Many proofs assume language is total.
  - No non-termination.
  - No bottoms.
  - Often also no infinite values.
- When this is not the case:
  Fast and loose reasoning.

Another example

- Program derived from specification using total methods.
- Result transcribed into partial language.

This work

Fast and loose reasoning is morally correct

Example

Total methods sometimes cheaper.

Example

$$reverse \circ map\ (\lambda x.x - y)$$

is the left inverse of

$$map\ (\lambda x.y + x) \circ reverse.$$

Example, total language

$$(\mathit{reverse} \circ \mathit{map}\ (\lambda x.x - y)) \circ (\mathit{map}\ (\lambda x.y + x) \circ \mathit{reverse})$$

$$= \ \{\mathit{map}\ f \circ \mathit{map}\ g = \mathit{map}\ (f \circ g), \circ \text{ associative}$$

$$\mathit{reverse} \circ \mathit{map}\ ((\lambda x.x - y) \circ (\lambda x.y + x)) \circ \mathit{reverse}$$

$$= \ \{(\lambda x.x - y) \circ (\lambda x.y + x) = \mathit{id}$$

$$\mathit{reverse} \circ \mathit{map}\ \mathit{id} \circ \mathit{reverse}$$

$$= \ \{\mathit{map}\ \mathit{id} = \mathit{id}, \circ \text{ associative}, \mathit{id} \circ f = f, \mathit{reverse} \circ \mathit{reverse} = \mathit{id}$$

$$\mathit{id}$$

Example, partial language

$$(\mathit{reverse} \circ \mathit{map}\ (\lambda x.x - y)) \circ (\mathit{map}\ (\lambda x.y + x) \circ \mathit{reverse})$$

$$= \{\mathit{map}\ f \circ \mathit{map}\ g = \mathit{map}\ (f \circ g), \circ \text{ associative}$$

$$\mathit{reverse} \circ \mathit{map}\ ((\lambda x.x - y) \circ (\lambda x.y + x)) \circ \mathit{reverse}$$

$$= \{\boxed{(\lambda x.x - y) \circ (\lambda x.y + x) = \mathit{id}}$$

$$\mathit{reverse} \circ \mathit{map}\ \mathit{id} \circ \mathit{reverse}$$

$$= \{\mathit{map}\ \mathit{id} = \mathit{id}, \circ \text{ associative}, \mathit{id} \circ f = f, \mathit{reverse} \circ \mathit{reverse} = \mathit{id}$$

$$\mathit{id}$$

Problem

$$\textbf{data}\ Nat = Zero \mid Succ\ Nat$$

$$\begin{tikzcd}[column sep=small]
 & & & & & infinity \\
 & & Succ\ (Succ\ Zero) \arrow[dr, dash] & & & \\
 & Succ\ Zero \arrow[dr, dash] & & Succ\ (Succ\ \perp) \arrow[dl, dash] \arrow[uurr, dash, dotted] & & \\
Zero \arrow[dr, dash] & & Succ\ \perp \arrow[dl, dash] & & & \\
 & \perp & & & &
\end{tikzcd}$$

Problem

$$(y + x) - y = x$$

- $(\mathit{Succ}\ \mathit{Zero} + \mathit{Succ}\ \perp) - \mathit{Succ}\ \mathit{Zero}$
  $= \perp \neq \mathit{Succ}\ \perp$
- $(\mathit{infinity} + \mathit{Zero}) - \mathit{infinity}$
  $= \perp \neq \mathit{Zero}$

Example, partial language

$$
\begin{aligned}
& (\mathit{reverse} \circ \mathit{map}\ (\lambda x.x - y)) \circ (\mathit{map}\ (\lambda x.y + x) \circ \mathit{reverse}) \\
= \quad & \{\mathit{map}\ f \circ \mathit{map}\ g = \mathit{map}\ (f \circ g), \circ \text{ associative} \\
& \mathit{reverse} \circ \mathit{map}\ ((\lambda x.x - y) \circ (\lambda x.y + x)) \circ \mathit{reverse} \\
= \quad & \{\boxed{(\lambda x.x - y) \circ (\lambda x.y + x) = \mathit{id}} \\
& \mathit{reverse} \circ \mathit{map}\ \mathit{id} \circ \mathit{reverse} \\
= \quad & \left\{ \begin{array}{l} \mathit{map}\ \mathit{id} = \mathit{id}, \circ \text{ associative}, \boxed{\mathit{id} \circ f = f}, \\ \boxed{\mathit{reverse} \circ \mathit{reverse} = \mathit{id}} \end{array} \right. \\
& \mathit{id}
\end{aligned}
$$

Example, partial language

Assume that $xs :: [Nat]$ and $y :: Nat$ are total, finite.

$$
\begin{aligned}
&((reverse \circ map\ (\lambda x.x - y)) \circ \\
&\quad (map\ (\lambda x.y + x) \circ reverse))\ xs \\
=\ & \{ map\ f \circ map\ g = map\ (f \circ g), \text{definition of } \circ \\
&reverse\ (map\ ((\lambda x.x - y) \circ (\lambda x.y + x))\ (reverse\ xs)) \\
=\ & \begin{cases} x, y \text{ total} \wedge y \text{ finite} \Rightarrow ((\lambda x.x - y) \circ (\lambda x.y + x))\ x = id\ x, \\ xs \text{ total, finite} \Rightarrow reverse\ xs \text{ total, finite}, \\ ys \text{ total} \wedge \forall \text{ total } x.\ f\ x = g\ x \Rightarrow map\ f\ ys = map\ g\ ys \end{cases} \\
&reverse\ (map\ id\ (reverse\ xs)) \\
=\ & \begin{cases} map\ id = id, id\ ys = ys, \\ xs \text{ total, finite} \Rightarrow reverse\ (reverse\ xs) = xs \end{cases} \\
&xs
\end{aligned}
$$

But. . .

- Programs syntactically identical.
- "Total subset" of partial semantics basically the same as total semantics.
- So we could just use the total result extended with some preconditions?

Rest of the talk

- Two languages.
- PER: Moral equality.
- Total equality implies moral equality.
- Translate moral equality.

Two higher-order, typed FPLs

- Same syntax.
- Total, set-theoretic: $\langle\!\langle t \rangle\!\rangle$.
- Partial, domain-theoretic: $[\![t]\!]$.
  - Pointed CPOs, lifted types, strict and non-strict.
- Recursive types (polynomial).
  - Inductive/coinductive types.
  - *fold*/*unfold*, but not *fix*.

Moral equality ($\sim$)

- PER on semantic domains of partial language.
- Defines the total values.
- Functions:
  - $$f \sim_{\sigma \to \tau} g \iff f \neq \bot \;\wedge\; g \neq \bot \;\wedge\; \forall x, y \in [\![\sigma]\!] .\; x \sim_\sigma y \;\Rightarrow\; f\ x \sim_\tau g\ y$$

Moral equality ($\sim$)

- PER on semantic domains of partial language.
- Defines the total values.
- Algebraic data types:
  - Defined.
  - Same constructor.
  - Arguments related.

Moral equality ($\sim$)

- PER on semantic domains of partial language.
- Defines the total values.
- Lists:
  - $[\,] \sim_{[\sigma]} [\,], \quad [\,] \not\sim_{[\sigma]} [x], \quad [\,] \not\sim_{[\sigma]} \perp$
  - $[x] \sim_{[\sigma]} [y] \quad \Leftrightarrow \quad x \sim_\sigma y$
  - $[x_1, x_2, \ldots] \sim_{[\sigma]} [y_1, y_2, \ldots] \Leftrightarrow$
    $x_1 \sim_\sigma y_1 \;\wedge\; x_2 \sim_\sigma y_2 \;\wedge\; \ldots$

Total equality implies moral equality

$$\langle\!\langle t_1 \rangle\!\rangle = \langle\!\langle t_2 \rangle\!\rangle \quad \Rightarrow \quad [\![t_1]\!] \sim [\![t_2]\!]$$

t_1, t_2	Closed terms.
$\langle\!\langle \cdot \rangle\!\rangle$	Total semantics.
$[\![\cdot]\!]$	Partial semantics.

Example revisited

$$\begin{aligned} lhs = {} & (reverse \circ map\ (\lambda x.x - 1)) \\ & \circ (map\ (\lambda x.1 + x) \circ reverse) \end{aligned}$$

- $\langle\!\langle lhs \rangle\!\rangle = \langle\!\langle id \rangle\!\rangle$
- $[\![lhs]\!] \sim [\![id]\!]$
- $\forall xs :: [\,Nat\,].$
  $$[\![xs]\!] \sim [\![xs]\!] \;\Rightarrow\; [\![lhs\ xs]\!] \sim [\![xs]\!]$$
- $\ldots$
- $\forall$ fin, tot $xs :: [\,Nat\,].\quad [\![lhs\ xs]\!] = [\![xs]\!]$

Discussion

- Fast and loose proofs OK (in a sense).
  - Polymorphism, stronger recursive types, type constructors.
  - Equational reasoning.

Discussion

- Sometimes partial reasoning is to be preferred.
  - Limited to total subset of the language.
  - Inductive and coinductive types separate:
    No hylomorphisms ($pretty \circ parse$).

Discussion

- Combining partial and total methods probably useful, but...
- ... $\sim$ is not a congruence,

$$x \sim y \quad \text{but} \quad f\ x \not\sim f\ y.$$

- Can translate, though.

Fast and loose reasoning is morally correct

Bicartesian closed category

…with initial algebras and final coalgebras.

- Objects: Types.
- Morphisms: Equivalence classes of total functions.
- $(\circ)$: The equivalence class of the underlying $(\circ)$.