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

\( \text{reverse} \circ \text{map} \ (\lambda x. x - y) \)

is the left inverse of

\( \text{map} \ (\lambda x. y + x) \circ \text{reverse}. \)
Example, total language

\[
(reverse \circ map (\lambda x. x - y)) \circ (map (\lambda x. y + x) \circ reverse)
\]

\[
= \{ map f \circ map g = map (f \circ g), \circ \text{associative} \}
\]

\[
reverse \circ map ((\lambda x. x - y) \circ (\lambda x. y + x)) \circ reverse
\]

\[
= \{ (\lambda x. x - y) \circ (\lambda x. y + x) = id \}
\]

\[
reverse \circ map id \circ reverse
\]

\[
= \{ map id = id, \circ \text{associative}, id \circ f = f, reverse \circ reverse = id \}
\]

id
Example, partial language

\[(\text{reverse} \circ \text{map} (\lambda x. x - y)) \circ (\text{map} (\lambda x. y + x) \circ \text{reverse})\]

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

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

\[= \{ (\lambda x. x - y) \circ (\lambda x. y + x) = \text{id} \}

\text{reverse} \circ \text{map id} \circ \text{reverse}

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

\text{id}
Problem

```
data Nat = Zero | Succ Nat
```

infinity

```
Succ (Succ Zero)  Succ (Succ ⊥)
  Succ Zero        Succ (Succ ⊥)
    Zero           Succ ⊥
                     ⊥
```
Problem

\[(y + x) - y = x\]

- \[(\text{Succ Zero} + \text{Succ } \bot) - \text{Succ Zero}\]
  \[= \bot \neq \text{Succ } \bot\]
- \[(\text{infinity} + \text{Zero}) - \text{infinity}\]
  \[= \bot \neq \text{Zero}\]
Example, partial language

\[
(reverse \circ map (\lambda x. x - y)) \circ (map (\lambda x. y + x) \circ reverse)
\]

\[
= \begin{cases} 
  \text{map } f \circ \text{map } g = \text{map } (f \circ g), \circ \text{associative} \\
\end{cases}
\]

\[
reverse \circ map ((\lambda x. x - y) \circ (\lambda x. y + x)) \circ reverse
\]

\[
= \begin{cases} 
  (\lambda x. x - y) \circ (\lambda x. y + x) = id \\
\end{cases}
\]

\[
reverse \circ map \ id \circ reverse
\]

\[
= \begin{cases} 
  \text{map } id = id, \circ \text{associative, } id \circ f = f, \\
  reverse \circ reverse = id \\
\end{cases}
\]

\[
id
\]
Example, partial language

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

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

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

$$\text{reverse} \ (\text{map} \ \text{id} \ (\text{reverse} \ xs))$$

$$= \begin{cases} \text{map} \ \text{id} = \text{id}, \ \text{id} \ ys = ys, \\
\quad xs \text{ total, finite } \Rightarrow \text{reverse} \ (\text{reverse} \ xs) = xs
\end{cases}
$$
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: $\langle t \rangle$.
- Partial, domain-theoretic: $\llbracket t \rrbracket$.
  - 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:
  - $f \sim_{\sigma \rightarrow \tau} g \iff f \neq \bot \land g \neq \bot \land \forall x, y \in \llbracket \sigma \rrbracket . x \sim_{\sigma} y \Rightarrow f x \sim_{\tau} g y$
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:
  - $[] \sim_{[\sigma]} [], [] \not\sim_{[\sigma]} [x], [] \not\sim_{[\sigma]} \bot$
  - $[x] \sim_{[\sigma]} [y] \iff x \sim_{\sigma} y$
  - $[x_1, x_2, \ldots] \sim_{[\sigma]} [y_1, y_2, \ldots] \iff x_1 \sim_{\sigma} y_1 \land x_2 \sim_{\sigma} y_2 \land \ldots$
Total equality implies moral equality

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

\( t_1, t_2 \) Closed terms.

\( \langle \cdot \rangle \) Total semantics.

\( [\cdot] \) Partial semantics.
Example revisited

\[
\text{lhs} = (\text{reverse} \circ \text{map} (\lambda x. x - 1)) \\
\circ (\text{map} (\lambda x. 1 + x) \circ \text{reverse})
\]

\[
\langle \langle \text{lhs} \rangle \rangle = \langle \langle \text{id} \rangle \rangle
\]

\[
\llbracket \text{lhs} \rrbracket \sim \llbracket \text{id} \rrbracket
\]

\[
\forall xs :: [Nat]. \quad \llbracket xs \rrbracket \sim \llbracket xs \rrbracket \Rightarrow \llbracket \text{lhs} \; xs \rrbracket \sim \llbracket xs \rrbracket
\]

\[
\ldots
\]

\[
\forall \text{fin, tot } xs :: [Nat]. \quad \llbracket \text{lhs} \; xs \rrbracket = \llbracket xs \rrbracket
\]
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 \((\text{pretty} \circ \text{parse})\).
Discussion

- Combining partial and total methods probably useful, but...

- \( \ldots \sim \) 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 \( (\circ) \).