Axiomatizing models of reversible computing

Robin Kaarsgaard (University of Copenhagen)
Mathys Rennela (Radboud University)

QCLS Seminar
February 1st, 2017

Outline

Reversible computing: What? Why?

Reversible programming primer

Join inverse category theory

Categorical semantics of Rfun

Where we are, sofar

Reversible computing: What? Why?

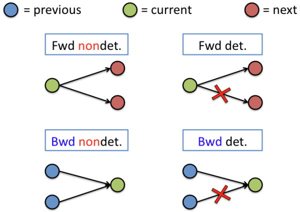
Reversible programming primer

Join inverse category theory

Categorical semantics of Rfun

What is reversible computing?

- Reversible computing: The study of *time invertible* computations.
- Deterministic in both forward and backward directions.



- Reversible functions are *injective*.
- *Totality* is not required in order to guarantee reversibility.

Why reversible computing?

- Originally motivated by the potential to reduce power consumption of computing processes.

- *Landauer's principle*: Irreversibility costs energy (Landauer, 1961).

- Plays a role in quantum computing and parallel computing.

- Example of reversible programming language: RFun.

Where we are, sofar

Reversible computing: What? Why?

Reversible programming primer

Join inverse category theory

Categorical semantics of Rfun

RFun

$$
\begin{aligned}
\mathit{fib}\ n \triangleq\ & \textbf{case}\ n\ \textbf{of} \\
& Z \quad\ \ \to \langle S(Z), S(Z)\rangle \\
& S(m) \to \textbf{let}\ \langle x, y\rangle = \mathit{fib}\ m\ \textbf{in} \\
& \qquad\quad\ \textbf{let}\ z = \mathit{plus}\ \langle y, x\rangle\ \textbf{in}\ z
\end{aligned}
$$

$$
\begin{aligned}
\mathit{plus}\ \langle x, y\rangle \triangleq\ & \textbf{case}\ y\ \textbf{of} \\
& Z \quad\ \to \lfloor\langle x\rangle\rfloor \\
& S(u) \to \textbf{let}\ \langle x', u'\rangle = \mathit{plus}\ \langle x, u\rangle\ \textbf{in}\ \langle x', S(u')\rangle
\end{aligned}
$$

- Untyped first-order reversible functional programming language.
- Patterns are linear: All variables defined by a pattern must be used *exactly once*.
- Results of all function calls must be bound in a `let`-expression.

RFun: linearity

- Linearity is essential!

- Explicit duplication via the duplication/equality operator $\lfloor \cdot \rfloor$

Invalid

$$dbl\ x \triangleq \mathbf{case}\ x\ \mathbf{of}\quad h : t \rightarrow h : h : t$$

Well-formed

$$\begin{aligned} dbl\ x \triangleq\ & \mathbf{case}\ x\ \mathbf{of} \\ & \quad h : t \rightarrow \mathbf{case}\ \lfloor \langle h \rangle \rfloor\ \mathbf{of} \\ & \qquad \langle h_1, h_2 \rangle \rightarrow h_1 : h_2 : t \end{aligned}$$

RFun: Recursion

- Recursion in RFun is based on a call stack, as in irreversible functional programming.
- Recursive functions are inverted by inverting the body of the `let`, and replacing the recursive call with a call to the inverse.

RFun: More restrictions

- Function and variable identifiers do not belong to the same sort.
- Programs = sequences of (function) definitions.
- Definitions must have (pairwise) distinct functional identifiers.
- In a left expression, a variable must appear exactly once.
- Domains of substitutions are (pairwise) disjoint.
- **Theorem [Yokoyama et al.]:**

  RFun can simulate any reversible Turing machine.

RFun: study of an example

$$
\begin{aligned}
&\mathit{fib}\ n \triangleq \textbf{case}\ n\ \textbf{of}\\
&\qquad Z \quad\ \ \to \langle S(Z), S(Z)\rangle\\
&\qquad S(m) \to \textbf{let}\ \langle x, y\rangle = \mathit{fib}\ m\ \textbf{in}\\
&\qquad\qquad\quad \textbf{let}\ z = \mathit{plus}\ \langle y, x\rangle\ \textbf{in}\ z
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{plus}\ \langle x, y\rangle \triangleq \textbf{case}\ y\ \textbf{of}\\
&\qquad Z \quad\ \ \to \lfloor\langle x\rangle\rfloor\\
&\qquad S(u) \to \textbf{let}\ \langle x', u'\rangle = \mathit{plus}\ \langle x, u\rangle\ \textbf{in}\ \langle x', S(u')\rangle
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{fib}^{-1}\ x_1 \triangleq \textbf{case}\ x_1\ \textbf{of}\\
&\qquad \langle S(Z), S(Z)\rangle \to Z\\
&\qquad x_2 \qquad\qquad\quad\ \to \textbf{let}\ \langle y, x\rangle = \mathit{plus}^{-1}\ x_2\ \textbf{in}\\
&\qquad\qquad\qquad\qquad\quad \textbf{let}\ m = \mathit{fib}^{-1}\ \langle x, y\rangle\ \textbf{in}\\
&\qquad\qquad\qquad\qquad\quad S(m)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{plus}^{-1}\ z \triangleq \textbf{case}\ z\ \textbf{of}\\
&\qquad \lfloor\langle x\rangle\rfloor \qquad\ \to \langle x, Z\rangle\\
&\qquad \langle x', S(u')\rangle \to \textbf{let}\ \langle x, u\rangle = \mathit{plus}^{-1}\ \langle x', u'\rangle\ \textbf{in}\ \langle x, S(u)\rangle
\end{aligned}
$$

Where we are, sofar

Reversible computing: What? Why?

Reversible programming primer

Join inverse category theory

Categorical semantics of Rfun

Inverse categories

- A restriction category is a category where each $f : A \to B$ has a *restriction idempotent* $\overline{f} : A \to A$ (subject to axioms such as $f \circ \overline{f} = f$, and others).
- Partial order enriched; for parallel morphisms f and g,

$$f \leq g \quad \text{iff} \quad g \circ \overline{f} = f$$

- *Partial isomorphism*: A morphism $f : A \to B$ with a partial inverse $f^\dagger : B \to A$ such that $f^\dagger \circ f = \overline{f}$ and $f \circ f^\dagger = \overline{f^\dagger}$.
- *Inverse category*: Restriction category with only partial isomorphisms.

Join inverse categories

An inverse category is a *join inverse category* if it has

- a *restriction zero*, specifically all *zero morphisms* $0_{A,B} : A \to B$,

- a partial operation $\bigvee$ on all *compatible* subsets of all hom-sets, satisfying

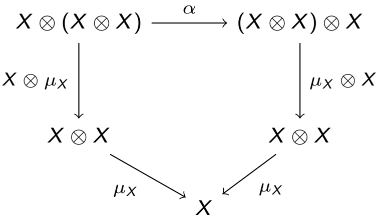
$$g \leq \bigvee_{f \in F} f \text{ if } g \in F, \text{ and if } f \leq h \text{ for all } f \in F \text{ then } \bigvee_{f \in F} f \leq h$$

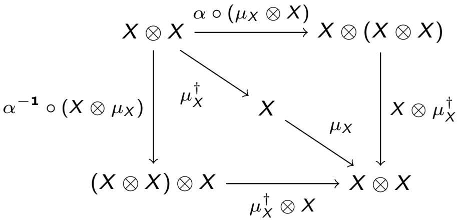
and other coherence axioms.

"Join inverse categories = inverse categories with *joins of countable homsets*"

The bimonoidal structure

In a (symmetric) monoidal †-category, a †-*Frobenius semialgebra* is a pair $(X, \mu_X : X \to X \otimes X)$ of an object X and a map μ_X such that the diagrams below commute.





Where we are, sofar

Reversible computing: What? Why?

Reversible programming primer

Join inverse category theory

Categorical semantics of Rfun

Denotational semantics

- Left expressions: $l ::= x \mid c(l_1, \ldots, l_n)$

- $\mathcal{S}$: denumerable object of symbols

- $T\mathcal{S}$: object of Rfun terms

- Rfun term as nonempty finite tree with values in $\mathcal{S}$

Categorical models of reversible computing

- A categorical model of reversible computing is a bimonoidal join inverse category (with decidable equality for $T\mathcal{S}$).

- Decidable equality: the equality of two elements is decidable (from a computability-theoretic PoV).

- Decidable equality for $T\mathcal{S}$ guarantees that there is a map $\mathrm{dupeq}_{T\mathcal{S}} : T\mathcal{S} \oplus (T\mathcal{S} \otimes T\mathcal{S}) \to T\mathcal{S} \oplus (T\mathcal{S} \otimes T\mathcal{S})$ to denote the duplication/equality operator.