Join inverse rig categories for reversible functional programming, and beyond

MFPS XXXVII

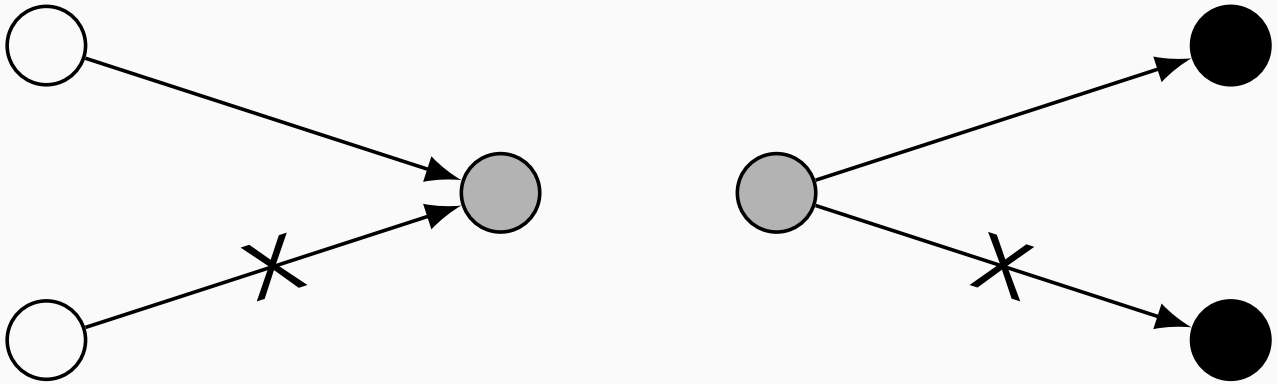
Robin Kaarsgaard[†] Mathys Rennela[‡]

September 1, 2021

[†] School of Informatics, University of Edinburgh
robin.kaarsgaard@ed.ac.uk

[‡] INRIA Paris
mathys.rennela@inria.fr

Reversible?



Reversible computation is the study of *forward and backward deterministic computation*.

As a consequence, reversible computations are invertible.

Rfun: Reversible functional programming

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

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

(Here, *fib* computes Fibonacci *pairs* while *plus* computes $\langle x, y\rangle \mapsto \langle x, x + y\rangle$.)

Untyped first-order reversible functional programming language with data in the form of LISP-style symbols and constructors (s-expressions).

Symmetric first match policy: Branches are tested in order given, but leaves of a branch must *not* match those produced by any branch above it.

Patterns are linear and variables must be used linearly as well (exactly once).

Duplication/equality: $\lfloor\langle x\rangle\rfloor = \langle x, x\rangle$, $\lfloor\langle x, x\rangle\rfloor = \langle x\rangle$, $\lfloor\langle x, y\rangle\rfloor = \langle x, y\rangle$

An inverse category is a category of *partial isomorphisms*.

In an inverse category, each morphism $A \xrightarrow{f} B$ is associated with a *restriction idempotent* $A \xrightarrow{\overline{f}} A$, thought of as a *partial identity defined where f is defined*, and a *partial inverse* $B \xrightarrow{f^\dagger} A$, such that

$$f^\dagger \circ f = \overline{f} \quad \text{and} \quad f \circ f^\dagger = \overline{f^\dagger} \ .$$

There are also some additional laws about restriction idempotents like $f \circ \overline{f} = f$, $\overline{f} \circ \overline{g} = \overline{g} \circ \overline{f}$, and more.

Examples: $\mathbf{PInj}$, $\mathbf{PHom}$, any groupoid, $\mathrm{Inv}(\mathbf{C})$ for a restriction category $\mathbf{C}$.

Inverse and rig categories

A rig category is a category equipped with two different symmetric monoidal products, such that one distributes over the other (up to isomorphism), subject to four pages worth of coherence conditions.

We usually write them as $(\otimes, I)$ and $(\oplus, O)$ by analogy with distributivity of multiplication over addition.

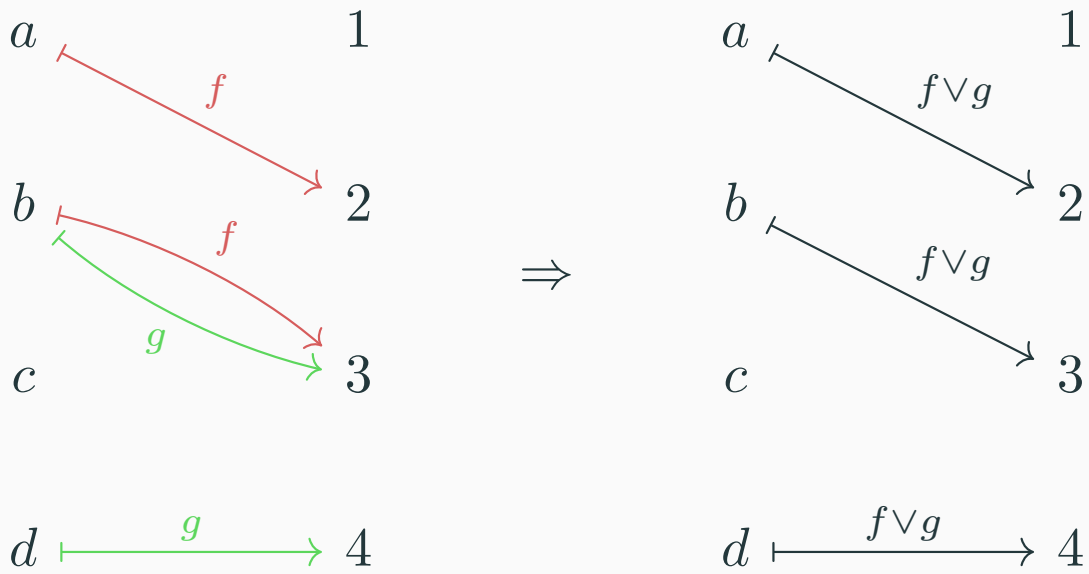
An inverse rig category is an inverse category with a rig structure (further satisfying some technical requirements, such as the presence of a natural diagonal $X \xrightarrow{\Delta} X \otimes X$ subject to some equations).

Examples: $\mathbf{PInj}$, $\mathrm{Inv}(\mathbf{C})$ for any distributive restriction category $\mathbf{C}$ (with discrete restriction products).

Non-examples: Non-trivial groupoids, $\mathbf{PHom}$ (but both due to the technical conditions).

Joins, by example

Let f and g be partial injective functions $A \to B$. Then $f \vee g$ exists if f and g agree in their overlap *in both the forward and backward directions*: $f(x) = g(x)$ whenever f and g both defined at x, and same for $f^\dagger$ and $g^\dagger$.



Note that maps which are entirely disjoint (i.e., in both domain of definition and image) are always compatible.

Joins, abstractly

An inverse category has joins if for all $A \xrightarrow{f,g} B$ with $f \circ \overline{g} = g \circ \overline{f}$ and $f^\dagger \circ \overline{g^\dagger} = g^\dagger \circ \overline{f^\dagger}$, $f \vee g$ exists (subject to some axioms).

An inverse category with joins of all sets of pairwise compatible morphisms is called a join inverse category.

A functor of join inverse categories is a functor that preserves joins.

Specifically, in a join inverse rig category, $\oplus$ and $\otimes$ preserve joins.

But what are they good for?

Man, they really tie the semantics together

Man, they really tie the semantics together

They allow us to define morphisms piecemeal, and then glue the pieces together to form the whole.

This is useful for

- recursively defined functions,
- case constructs,
- duplication/equality,

and more!

To construct an object of values, we assume that the following two fixed points exist (see paper):

$$L(A) = \mu X.I \oplus (A \otimes X) \qquad T(A) = \mu X.A \otimes L(X)$$

We further assume that we're given an object S of *symbols*. Then we construct interpretations of

- open (left) expressions of n free variables as morphisms $T(S)^{\otimes n} \to T(S)$,
- patterns of n variables as morphisms $T(S) \to T(S)^{\otimes n}$,
- functions as open expressions of a single free variable.

Note in particular that this encoding enforces the linear use of variables.

Pattern matching is dual to value construction

The interpretation of an open left expression can be seen as a *value constructor*: They construct values by taking appropriate amount of values to fill out the holes left by free variables.

In **PInj**,

$$[\![\langle x, y\rangle]\!] = (v_1, v_2) \mapsto \langle v_1, v_2\rangle$$

Notice that the partial inverse to this value constructor performs *pattern matching*. Given a value v,

- if $v = \langle v_1, v_2\rangle$ then $[\![\langle x, y\rangle]\!]^\dagger(v) = (v_1, v_2)$ (corresponding to binding x to v_1 and y to v_2); and
- if v is not of this form then $[\![\langle x, y\rangle]\!]^\dagger(v)$ is undefined.

For this reason, the interpretation of patterns is given by the *partial inverse to their interpretations as open left expressions*.

Case expressions

Finally, we use joins to construct the interpretation of case expressions.

To do this, we interpret branches as separate morphisms, and then glue them together with joins.

Problem: How do we know that the resulting morphisms are compatible?

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

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

The symmetric first match policy ensures compatibility, if we can ensure that branches are tested in the given order.

Decidable pattern matching

When $T(S) \xrightarrow{[\![l]\!]^\dagger} T(S)^{\otimes n}$ is the interpretation of a pattern l, its restriction idempotent $\overline{[\![l]\!]^\dagger}$ is a partial identity defined only on values which *match* that pattern.

To be able to enforce order in pattern matching, we need that all such restriction idempotents have disjoint complements, satisfying $\overline{[\![l]\!]^\dagger} \vee \overline{[\![l]\!]^\dagger}^\perp = \mathrm{id}$.

We call this property *decidable pattern matching*, and we require it in order to give semantics to case expressions with the symmetric first match policy.

Case expressions, again

We give an interpretation of a case expression such as

$$\begin{aligned}&\textbf{case } l \textbf{ of}\\ &\qquad l_1 \to e_1\\ &\qquad l_2 \to e_2\\ &\qquad l_3 \to e_3\end{aligned}$$

(ignoring l for now) branch by branch, as in

1. $[\![e_1]\!] \circ [\![l_1]\!]^\dagger \quad \vee$
2. $[\![e_2]\!] \circ [\![l_2]\!]^\dagger \circ \overline{[\![l_1]\!]^\dagger}^\perp \quad \vee$
3. $[\![e_3]\!] \circ [\![l_3]\!]^\dagger \circ \overline{[\![l_2]\!]^\dagger}^\perp \circ \overline{[\![l_1]\!]^\dagger}^\perp$

and so on.

Compare this to Gram-Schmidt orthogonalization.

While the focus here is on a semantics for Rfun, our approach is powerful enough to accomodate other languages. We rely on this theorem:

Theorem: Any join inverse category $\mathbf{C}$ with a disjointness tensor (specifically any join inverse rig category) is equipped with a uniform dagger trace $\mathbf{C}(A \oplus U, B \oplus U) \to \mathbf{C}(A, B)$.

- Join inverse categories with a disjointness tensor, specifically join inverse rig categories, form models of reversible flowcharts.
- Dagger traced ω-continuous rig categories, including those join inverse rig categories which are ω-complete for join preserving functors, form models of $\Pi^0 \Rightarrow$ Theseus.

Concluding remarks

Reversible computing is a novel computing paradigm with applications in areas such as low-power computing, quantum computing, and even robotics.

Join inverse rig categories is a powerful framework for modelling reversible programming languages.

Stay tuned for more reversible content this session!

Thank you!

Duplication/equality

The interpretation of the duplication/equality operator is constructed as the join of three disjoint morphisms $T(S) \to T(S)$.

1. One which sends the "unary tuple" $\langle x \rangle$ to the pair $\langle x, x \rangle$,
2. one which sends a pair of equal things $\langle x, x \rangle$ to $\langle x \rangle$, and
3. one which, when $x \neq y$, sends $\langle x, y \rangle$ to $\langle x, y \rangle$ unchanged.

The first is constructed using the natural diagonal $T(S) \xrightarrow{\Delta} T(S) \otimes T(S)$, and the second by its partial inverse $T(S) \otimes T(S) \xrightarrow{\Delta^\dagger} T(S)$.

Notice that $T(S) \otimes T(S) \xrightarrow{\Delta^\dagger} T(S)$ is only defined on pairs of equal things, so its restriction idempotent $\overline{\Delta^\dagger}$ is a partial identity defined only on pairs of equal values.

Decidable equality

To construct the third map, we need a *complement* to this restriction idempotent; a restriction idempotent

$$T(S) \otimes T(S) \xrightarrow{\overline{\Delta^\dagger}^\perp} T(S) \otimes T(S)$$

defined only on pairs of *distinct* values, i.e., satisfying $\overline{\Delta^\dagger} \vee \overline{\Delta^\dagger}^\perp = \mathrm{id}$. We call this property *decidable equality*, and we require it in order to construct an interpretation of duplication/equality.