JOIN INVERSE CATEGORIES AND REVERSIBLE RECURSION

NORDIC WORKSHOP ON PROGRAMMING THEORY 2015

Robin Kaarsgaard
October 21, 2015

DIKU, Department of Computer Science, University of Copenhagen
robin@di.ku.dk
http://www.di.ku.dk/~robin
WHO?

- Robin Kaarsgaard, PhD student at DIKU, Dept. of Computer Science, University of Copenhagen.

2. Reversible functional programming
   - RFUN
   - Theseus (and \( \Pi^0 \))

3. Join inverse categories and reversible recursion

4. Concluding remarks
REVERSIBLE COMPUTING: WHAT? WHY?
Reversible computing: The study of time invertible computations.

Deterministic in both forward and backward directions.

In a functional programming setting, reversible functions are injective.

Note that totality is not required, nor necessarily desirable, in order to guarantee reversibility.
• Originally motivated by the potential to reduce power consumption of computing processes, due to Landauer’s principle: Irreversibility costs energy.
• Has since seen a number of applications independent of this property; personal favorites include
  • unified parser/pretty printer specifications and
  • fast parallel discrete event simulations.
• Plays an important role in quantum computing.


REVERSIBLE FUNCTIONAL PROGRAMMING
\begin{align*}
\text{fib } n & \triangleq \text{case } n \text{ of} \\
Z & \rightarrow \langle S(Z), S(Z) \rangle \\
S(m) & \rightarrow \text{let } \langle x, y \rangle = \text{fib } m \text{ in} \\
& \quad \text{let } z = \text{plus } \langle y, x \rangle \text{ in } z \\
\text{plus } \langle x, y \rangle & \triangleq \text{case } y \text{ of} \\
Z & \rightarrow \lfloor \langle x \rangle \rfloor \\
S(u) & \rightarrow \text{let } \langle x', u' \rangle = \text{plus } \langle x, u \rangle \text{ in } \langle x', S(u') \rangle
\end{align*}

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

---

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.

---

Another choice, similar to that made by Janus, is to use bounded integers such that every operation is always well defined through underflows and overflows. Here is a simple 4-bit type Nat4:

```
<table>
<thead>
<tr>
<th>add1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nat4</td>
</tr>
<tr>
<td>Nat4</td>
</tr>
<tr>
<td>Nat4</td>
</tr>
<tr>
<td>Nat4</td>
</tr>
</tbody>
</table>
```


\[ \text{treeUnwindf} :: f : (\text{Nat} \leftrightarrow a) \rightarrow \text{Tree} \leftrightarrow \text{Tree} \times \text{Tree} + a \]

| Node t1 t2 \leftrightarrow Left (t1, t2) |
| Leaf n \leftrightarrow Right (f n) |

- Typed first-order reversible functional programming language
- Supports parametrized maps, maps depending on other maps given at compile time.
- Patterns are linear and exhaustive, all functions are total.
- Compiles to the reversible combinator calculus \( \Pi^0 \).

---


Recursion in Theseus (indirectly) and $\Pi^0$ (directly) is implemented via a reversible trace operator

\[ \text{trace} : a + x \leftrightarrow b + x \rightarrow a \leftrightarrow b \]

This is a trace in the categorical sense of traced monoidal categories (in fact, a $\dagger$-trace).

---


THESEUS AND $\Pi^0$: RECUSION VIA $\dagger$-TRACE

- Recursion in Theseus (indirectly) and $\Pi^0$ (directly) is implemented via a reversible trace operator

$$\text{trace} : a + x \leftrightarrow b + x \rightarrow a \leftrightarrow b$$

- This is a trace in the categorical sense of traced monoidal categories (in fact, a $\dagger$-trace).

---


JOIN INVERSE CATEGORIES AND REVERSIBLE RECURSION
• **Wanted:** Categorical model rich enough to capture...
  • *partial* injective functions (*RFUN* isn’t total), and
  • the two *distinct* notions of reversible recursion from *RFUN* and *Theseus*

• Starting point: Giles’ investigation of inverse categories as models of reversible functional programming.

• Inverse categories: Special case of restriction categories, categories with partiality.

---

• 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 \( g \),

\[
f \leq g \iff g \circ \overline{f} = f
\]

• 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.

---

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 axioms.
- We consider inverse categories with **joins of countable sets**.

---

• **Observation**: The underlying sets for all $\omega$-chains are compatible.

• **Idea**: Given an $\omega$-chain $\{f_i\}_{i \in \omega}$, define $\sup \{f_i\}_{i \in \omega} = \bigvee_{i \in \omega} f_i$.

• **Consequence (by Kleene’s fixed point theorem)**: Every monotone and continuous morphism scheme of the form $f : \text{Hom}_C(A, B) \to \text{Hom}_C(A, B)$ has a least fixed point $\text{fix} f : A \to B$.
  
  • Morphism schemes in general look a whole lot like parametrized maps à la Theseus...
• **Insight**: The family of morphism schemes defined by $\text{inv}_{A,B}(f) = f^\dagger$ is monotone, continuous, and an isomorphism with inverse $\text{inv}_{B,A}$ in each component.

• Every monotone and continuous morphism scheme of the form $f : \text{Hom}_\mathcal{C}(A, B) \to \text{Hom}_\mathcal{C}(A, B)$ has a *fixed point adjoint* $f^\dagger : \text{Hom}_\mathcal{C}(B, A) \to \text{Hom}_\mathcal{C}(B, A)$ such that $(\text{fix } f)^\dagger = \text{fix } f^\dagger$.
  
  • Trick: Define $f^\dagger = \text{inv}_{A,B} \circ f \circ \text{inv}_{B,A}$.

• This is precisely recursion à la RFUN!
• Unique decomposition categories (UDCs) are categories with...
  • a partial sum operator $\Sigma$ on countable families of parallel morphisms, and
  • a sum-like monoidal tensor $\cdot \oplus \cdot$

both subject to certain axioms.

• **Result** (Haghverdi): Given the existence of certain sums, UDCs have a (uniform) trace.

• **Idea**: Define $\sum_{i \in I} f_i = \bigvee_{i \in I} f_i$, and get the sum-like monoidal tensor via a join-preserving *disjointness tensor* (Giles).

---


• **Result:** Not just a trace operator, but one satisfying the $\dagger$-trace condition

$$\text{Tr}^{X,Y}_{A,B}(f)^\dagger = \text{Tr}^{X,Y}_{B,A}(f^\dagger)$$

for all $f : A \oplus X \to B \oplus X$.

• Reversible recursion à la Theseus and $\Pi^0$!

---

CONCLUDING REMARKS
• All of the gory details!
  • A few more are in the abstract – for the rest, just ask!
• Using Adámek’s fixed point theorem, Guo’s join completion theorem, and a few lemmas, we can also show faithful embedding in algebraically $\omega$-compact category: This models isorecursive data types à la Theseus.
• By viewing join inverse categories as CPO-categories, we get
  • fixed points of morphism schemes, modelling reversible recursion à la RFUN.

• Additionally assuming the existence of a join-preserving disjointness tensor, we get
  • a $\dagger$-trace operator for modelling reversible tail recursion à la Theseus and $\Pi^0$.

• Next up:
  • Use these insights to inform language design.
  • Compact closed inverse categories – relation to partiality in quantum computing?
  • Suggestions? Talk to me!
Thank you!