Dagger Traced Symmetric Monoidal Categories and Reversible Programming

William J. Bowman, Roshan P. James, Amr Sabry

27th June, 2011

Outline

- Goal: Teach you the language Π^o and show you some example programs in 15mins.

- We will skip much of the category theory. (Paper has details).

Introduction

- Π^o is reversible by design – all expressible programs are reversible.
- Does not start with an irreversible model and then "add history."
- Has a very simple definition in terms of isomorphisms between types.

Intuitions from arithmetic

- We know various equalities from arithmetic:
  - $n + 0 = n$
  - $n + m = m + n$
  - $n \times m = m \times n$
  - $n \times (m + l) = n \times m + n \times l$
  - $\dfrac{n = m \quad m = l}{n = l}$
  - $\dfrac{n + m = l + m}{n = l}$

Intuitions from arithmetic

- We know various equalities from arithmetic:
  - $n + 0 = n$
  - $n + m = m + n$
  - $n \times m = m \times n$
  - $n \times (m + l) = n \times m + n \times l$
  - $\dfrac{n = m \quad m = l}{n = l}$
  - $\dfrac{n + m = l + m}{n = l}$
- We build a language by treating these equalities as isomorphisms between types.

Intuitions from arithmetic

- We know various equalities from arithmetic:
  - $n + 0 = n$
  - $n + m = m + n$
  - $n \times m = m \times n$
  - $n \times (m + l) = n \times m + n \times l$
  - $$\frac{n = m \quad m = l}{n = l}$$
  - $$\frac{n + m = l + m}{n = l}$$
- We build a language by treating these equalities as isomorphisms between types.
- We will give them a computational interpretation.

Type Isomorphisms

Sound and Complete Isomorphisms:

$$\begin{array}{rcl} \textit{base types}, b & ::= & 0 \mid 1 \mid b + b \mid b \times b \\ \textit{values}, v & ::= & () \mid \textit{left } v \mid \textit{right } v \mid (v, v) \end{array}$$

$$\begin{array}{rclr} 0 + b & \leftrightarrow & b & \textit{identity for } + \\ b_1 + b_2 & \leftrightarrow & b_2 + b_1 & \textit{commutativity for } + \\ b_1 + (b_2 + b_3) & \leftrightarrow & (b_1 + b_2) + b_3 & \textit{associativity for } + \\ \\ 1 \times b & \leftrightarrow & b & \textit{identity for } \times \\ b_1 \times b_2 & \leftrightarrow & b_2 \times b_1 & \textit{commutativity for } \times \\ b_1 \times (b_2 \times b_3) & \leftrightarrow & (b_1 \times b_2) \times b_3 & \textit{associativity for } \times \\ \\ 0 \times b & \leftrightarrow & 0 & \textit{distribute over } 0 \\ (b_1 + b_2) \times b_3 & \leftrightarrow & (b_1 \times b_3) + (b_2 \times b_3) & \textit{distribute over } + \end{array}$$

$$\frac{}{b \leftrightarrow b} \qquad \frac{b_1 \leftrightarrow b_2}{b_2 \leftrightarrow b_1} \qquad \frac{b_1 \leftrightarrow b_2 \quad b_2 \leftrightarrow b_3}{b_1 \leftrightarrow b_3}$$

$$\frac{b_1 \leftrightarrow b_3 \quad b_2 \leftrightarrow b_4}{(b_1 + b_2) \leftrightarrow (b_3 + b_4)} \qquad \frac{b_1 \leftrightarrow b_3 \quad b_2 \leftrightarrow b_4}{(b_1 \times b_2) \leftrightarrow (b_3 \times b_4)}$$

Type Isomorphisms, Recursive Types and Trace

$$\begin{array}{rcl} \textit{base types}, b & ::= & 0 \mid 1 \mid b + b \mid b \times b \mid x \mid \mu x.b \\ \textit{values}, v & ::= & () \mid \textit{left } v \mid \textit{right } v \mid (v, v) \mid \langle v \rangle \end{array}$$

$$\begin{array}{rclr}
\mu x.b & \leftrightarrow & b[\mu x.b/x] & \textit{isorecursive types} \\
0 + b & \leftrightarrow & b & \textit{identity for } + \\
b_1 + b_2 & \leftrightarrow & b_2 + b_1 & \textit{commutativity for } + \\
b_1 + (b_2 + b_3) & \leftrightarrow & (b_1 + b_2) + b_3 & \textit{associativity for } + \\
1 \times b & \leftrightarrow & b & \textit{identity for } \times \\
b_1 \times b_2 & \leftrightarrow & b_2 \times b_1 & \textit{commutativity for } \times \\
b_1 \times (b_2 \times b_3) & \leftrightarrow & (b_1 \times b_2) \times b_3 & \textit{associativity for } \times \\
0 \times b & \leftrightarrow & 0 & \textit{distribute over } 0 \\
(b_1 + b_2) \times b_3 & \leftrightarrow & (b_1 \times b_3) + (b_2 \times b_3) & \textit{distribute over } +
\end{array}$$

$$\frac{}{b \leftrightarrow b} \qquad \frac{b_1 \leftrightarrow b_2}{b_2 \leftrightarrow b_1} \qquad \frac{b_1 \leftrightarrow b_2 \quad b_2 \leftrightarrow b_3}{b_1 \leftrightarrow b_3}$$

$$\frac{b_1 \leftrightarrow b_3 \quad b_2 \leftrightarrow b_4}{(b_1 + b_2) \leftrightarrow (b_3 + b_4)} \qquad \frac{b_1 \leftrightarrow b_3 \quad b_2 \leftrightarrow b_4}{(b_1 \times b_2) \leftrightarrow (b_3 \times b_4)} \qquad \frac{b_1 + b_2 \leftrightarrow b_1 + b_3}{b_2 \leftrightarrow b_3}$$

Witnesses for Type Isomorphisms

Primitive operators and their composition:

$$\begin{array}{rcl} \textit{base types}, b & ::= & 0 \mid 1 \mid b + b \mid b \times b \mid x \mid \mu x.b \\ \textit{values}, v & ::= & () \mid \textit{left } v \mid \textit{right } v \mid (v, v) \mid \langle v \rangle \end{array}$$

$$\begin{array}{rrcll} \textit{unfold}: & \mu x.b & \leftrightarrow & b[\mu x.b/x] & : \textit{fold} \\ \\ \succcurlyeq_+ : & 0 + b & \leftrightarrow & b & : \preccurlyeq_+ \\ \times_+ : & b_1 + b_2 & \leftrightarrow & b_2 + b_1 & : \times_+ \\ \gtrless_+ : & b_1 + (b_2 + b_3) & \leftrightarrow & (b_1 + b_2) + b_3 & : \lessgtr_+ \\ \\ \succcurlyeq_\times : & 1 \times b & \leftrightarrow & b & : \preccurlyeq_\times \\ \times_\times : & b_1 \times b_2 & \leftrightarrow & b_2 \times b_1 & : \times_\times \\ \gtrless_\times : & b_1 \times (b_2 \times b_3) & \leftrightarrow & (b_1 \times b_2) \times b_3 & : \lessgtr_\times \\ \\ \prec_0 : & 0 \times b & \leftrightarrow & 0 & : \succ_0 \\ \prec : & (b_1 + b_2) \times b_3 & \leftrightarrow & (b_1 \times b_3) + (b_2 \times b_3) & : \succ \end{array}$$

$$\frac{}{id : b \leftrightarrow b} \qquad \frac{c : b_1 \leftrightarrow b_2}{sym\ c : b_2 \leftrightarrow b_1} \qquad \frac{c_1 : b_1 \leftrightarrow b_2 \quad c_2 : b_2 \leftrightarrow b_3}{(c_1 \bullet c_2) : b_1 \leftrightarrow b_3}$$

$$\frac{c_1 : b_1 \leftrightarrow b_3 \quad c_2 : b_2 \leftrightarrow b_4}{(c_1 \oplus c_2) : (b_1 + b_2) \leftrightarrow (b_3 + b_4)} \qquad \frac{c_1 : b_1 \leftrightarrow b_3 \quad c_2 : b_2 \leftrightarrow b_4}{(c_1 \otimes c_2) : (b_1 \times b_2) \leftrightarrow (b_3 \times b_4)} \qquad \frac{c : b_1 + b_2 \leftrightarrow b_1 + b_3}{trace\ c : b_2 \leftrightarrow b_3}$$

The language Π^o

A programming language emerges:

Syntax

$$
\begin{array}{rcl}
\textit{base types}, b & ::= & 0 \mid 1 \mid b + b \mid b \times b \mid x \mid \mu x.b \\
\textit{values}, v & ::= & () \mid \textit{left } v \mid \textit{right } v \mid (v, v) \mid \langle v \rangle \\
\\
\textit{isomorphisms}, iso & ::= & \times_+ \mid \gtrsim_+ \mid \lesssim_+ \mid \succcurlyeq_+ \mid \preccurlyeq_+ \\
 & \mid & \times_\times \mid \gtrsim_\times \mid \lesssim_\times \mid \succcurlyeq_\times \mid \preccurlyeq_\times \\
 & \mid & \prec_0 \mid \succ_0 \mid \prec \mid \succ \mid \textit{id} \mid \textit{fold} \mid \textit{unfold} \\
\textit{combinators}, c & ::= & \textit{iso} \mid \textit{sym } c \mid c \bullet c \mid c \otimes c \mid c \oplus c \mid \textit{trace } c
\end{array}
$$

- Π^o is a dagger symmetric monoidal category with a trace over the monoid $(+, 0)$.
- In practice, we use "wiring diagrams" to program : values are like particles moving through a circuit.

Operational semantics: $c\ v_1 \mapsto v_2$

Programs c are applied to values v, thus evaluation is denoted by $c\ v_1 \mapsto v_2$ transitions.

- Associativity

$$\begin{array}{llcl} \gtrless_\times : & b_1 \times (b_2 \times b_3) & \leftrightarrow & (b_1 \times b_2) \times b_3 \\ \gtrless_\times & (v_1, (v_2, v_3)) & \mapsto & ((v_1, v_2), v_3) \end{array}$$

- Distribution

$$\begin{array}{llcl} \prec : & (b_1 + b_2) \times b_3 & \leftrightarrow & (b_1 \times b_3) + (b_2 \times b_3) \\ \prec & (\mathit{left}\ v_1, v_3) & \mapsto & \mathit{left}\ (v_1, v_3) \\ \prec & (\mathit{right}\ v_2, v_3) & \mapsto & \mathit{right}\ (v_2, v_3) \end{array}$$

- Folding

$$\begin{array}{llcl} \mathit{fold} : & b[\mu x.b/x] & \leftrightarrow & \mu x.b \\ \mathit{fold} & v & \mapsto & \langle v \rangle \end{array}$$

Traces give us Iteration

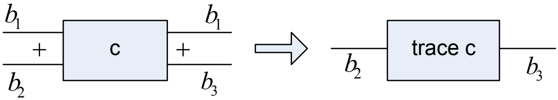
Categorical "trace" over the monoid $(+, 0)$ – sometimes called a "Cancellation Law."

$$\frac{c : b_1 + b_2 \leftrightarrow b_1 + b_3}{trace\ c : b_2 \leftrightarrow b_3}$$

Traces give us Iteration

Categorical "trace" over the monoid $(+, 0)$ – sometimes called a "Cancellation Law."

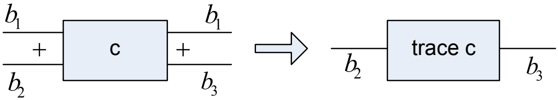
$$\frac{c : b_1 + b_2 \leftrightarrow b_1 + b_3}{trace\ c : b_2 \leftrightarrow b_3}$$

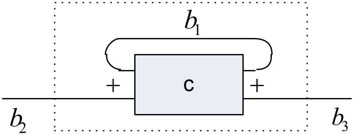


Traces give us Iteration

Categorical "trace" over the monoid $(+, 0)$ – sometimes called a "Cancellation Law."

$$\frac{c : b_1 + b_2 \leftrightarrow b_1 + b_3}{trace\ c : b_2 \leftrightarrow b_3}$$





The iterative interpretation satisfies the coherence conditions for categorical traces.

Programming in Π^o : Booleans, not

- $bool = 1 + 1$
- $true = left\ ()$
  $false = right\ ()$

Programming in Π^o : Booleans, not

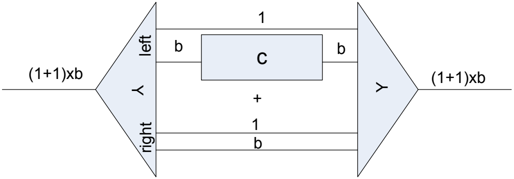
- $bool = 1 + 1$
- $true = left\ ()$
  $false = right\ ()$
- $not : bool \leftrightarrow bool$
  $not = \times_+$
- Verify the behavior:
  $not\ true \mapsto^* false$
  $not\ false \mapsto^* true$

Conditionals, Toffoli ...

One armed-if :

$if_c : bool \times b \leftrightarrow bool \times b$

$if_c\ (flag, b) = if\ flag\ then\ (flag, c(b))\ else\ (flag, b)$



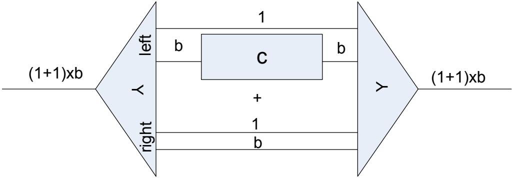
$if_c = \prec \bullet ((id \otimes c) \oplus id) \bullet \succ$

Conditionals, Toffoli ...

One armed-if :

$if_c : bool \times b \leftrightarrow bool \times b$

$if_c\ (flag, b) = if\ flag\ then\ (flag, c(b))\ else\ (flag, b)$



$if_c = \prec \bullet ((id \otimes c) \oplus id) \bullet \succ$

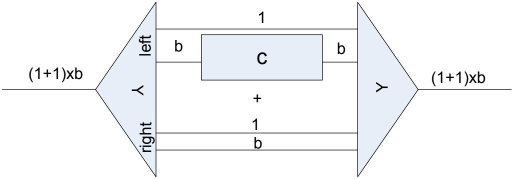
- $cnot : bool \times bool \leftrightarrow bool \times bool$
  $cnot = if_{not}$

Conditionals, Toffoli ...

One armed-if :

$if_c : bool \times b \leftrightarrow bool \times b$

$if_c\ (flag, b) = if\ flag\ then\ (flag, c(b))\ else\ (flag, b)$



$if_c = \prec \bullet ((id \otimes c) \oplus id) \bullet \succ$

- $cnot : bool \times bool \leftrightarrow bool \times bool$
  $cnot = if_{not}$
- $toffoli : bool \times (bool \times bool) \leftrightarrow bool \times (bool \times bool)$
  $toffoli = if_{cnot}$

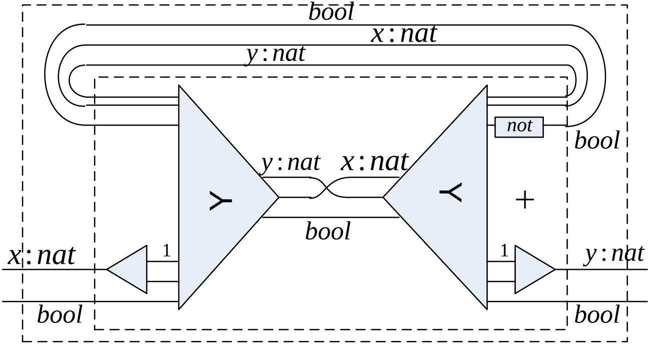
Iteration over numbers

- $nat = \mu x.1 + x$
- $0 = \langle left\ ()\rangle, 1 = \langle right\ 0\rangle, 2 = \langle right\ 1\rangle$...
- $fold : 1 + nat \leftrightarrow nat : unfold$

Iteration over numbers

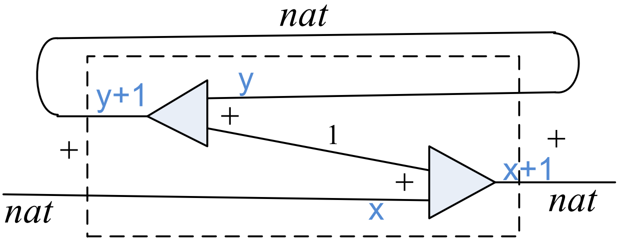
- $nat = \mu x.1 + x$
- $0 = \langle left\ ()\rangle, 1 = \langle right\ 0\rangle, 2 = \langle right\ 1\rangle$...
- $fold : 1 + nat \leftrightarrow nat : unfold$

We can construct a "for" loop:



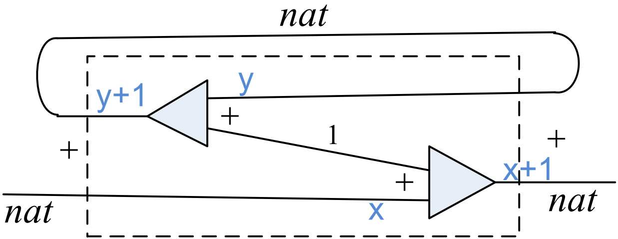
Non-termination and *inc* : $nat \leftrightarrow nat$

- $nat = \mu x.1 + x$
- $0 = \langle left\ ()\rangle, 1 = \langle right\ 0\rangle, 2 = \langle right\ 1\rangle ...$
- $fold : 1 + nat \leftrightarrow nat : unfold$



Non-termination and $inc : nat \leftrightarrow nat$

- $nat = \mu x.1 + x$
- $0 = \langle left\ ()\rangle, 1 = \langle right\ 0\rangle, 2 = \langle right\ 1\rangle ...$
- $fold : 1 + nat \leftrightarrow nat : unfold$



$$inc\ n \mapsto (n+1) \qquad inc^{-1}\ (n+1) \mapsto n$$
$$inc^{-1}\ 0 \mapsto undefined$$

- Π^o can express infinite loops.

Summary

- Introduced Π^o.
- The code accompanying the paper has many examples.
  - Factorial
  - Operations on lists.