Computing with Isomorphisms

Roshan P. James and Amr Sabry

School of Informatics and Computing
Indiana University.
rpjames@indiana.edu

26th June, 2012

Quote

The Fabric of Reality, David Deutsch (1997):

> *"Turing hoped that his abstracted-paper-tape model was so simple, so transparent and well defined, that it would not depend on any assumptions about physics that could conceivably be falsified, and therefore that it could become the basis of an abstract theory of computation that was independent of the underlying physics. 'He thought,' as Feynman once put it, 'that he understood paper.' But he was mistaken. Real, quantum-mechanical paper is wildly different from the abstract stuff that the Turing machine uses. The Turing machine is entirely classical..."*

Computing with the Physical World

- Quantum Physics suggests a universe in which
  - fundamental interactions are all reversible.
  - various quantities such as mass, energy, angular momentum, spin etc are conserved.

Computing with the Physical World

- Abstract models of computation shield us from the underlying technology that realize computation in the real world.
- Irreversible classical worldview.
  - Logic gates such as **nand** cannot recover inputs from outputs.
  - Turing machines rely on overwriting of symbols on a tape.
  - λ-calculus relies on β reduction.

Closed Systems and Interactions as Effects

- Closed systems are the basic notion and the basic unit of study in Physics.
  - Closed systems are reversible and obey various conservation laws.

- Open systems, ones that interact with their environment, are a derived notion.
  - Open systems do not share these properties.

Interactions as Effects

Interactions with the environment that change the properties of a closed system, are much like side effects .

Embedding Irreversibility within Reversibility

Toffoli (1980)

For every finite function $\phi : bool^m \rightarrow bool^n$ there exists an invertible finite function $\phi^R : bool^{r+m} \rightarrow bool^{r+m}$, with $r \leq n$, such that $\phi(x_1, \ldots, x_m) = (y_1, \ldots, y_n)$ iff

$$\phi^R(x_1, \ldots, x_m, \overbrace{false, \ldots, false}^{r}) = (\overbrace{\ldots}^{m+r-n}, y_1, \ldots, y_n)$$

Compiling AND

- Consider the irreversible function $and : bool \times bool \rightarrow bool$
- According to Toffoli (1980), "*and*" can be compiled to

F	*F*	F	|	F	F	*F*
T	*F*	F	|	T	F	*F*
F	*T*	F	|	F	T	*F*
T	*T*	F	|	T	T	*T*
F	F	T	|	F	F	T
T	F	T	|	T	F	T
F	T	T	|	F	T	T
T	T	T	|	T	T	F

$\boxed{\text{Thesis: Information Effects}}$

Thesis

By embodying irreversible physical primitives, conventional abstract models of computation have also inadvertently included some *implicit* computational effects, which we call *information effects*.

Thesis

By embodying irreversible physical primitives, conventional abstract models of computation have also inadvertently included some *implicit* computational effects, which we call *information effects*.

Technically:

1. A function is information preserving if its input entropy matches its output entropy.
2. Logically reversible functions are information preserving.
3. Irreversibility is a derived concept and is captured by a type-and-effect system.
   - Information effects are modeled as interactions with a surrounding information environment.

Contributions

- We develop such a “information preserving” reversible computational model called Π^o (also a strong normalizing fragment called Π).
- Reversibility is captured by (partial) type isomorphisms.
- Interactions with the heap and garbage are effects that are tracked by the type system.

$\boxed{\text{Computing with Isomorphisms}}$

Π^o : a semantic foundation for computation

- Π^o is based on
  1. type isomorphisms.
  2. trace operators from category theory.

Π^o : a semantic foundation for computation

- Π^o is based on
  1. type isomorphisms.
  2. trace operators from category theory.

- Information can neither be created nor deleted in Π^o much like
  - Restrictions on weakening and contraction in Linear Logic
  - The no-cloning and no-deletion theorems of Quantum Mechanics.

Type Isomorphisms for Finite types

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

Type Isomorphisms for Finite types

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

$$\begin{array}{rclr} 0 + b & \rightleftharpoons & b & \textit{identity for } + \\ b_1 + b_2 & \rightleftharpoons & b_2 + b_1 & \textit{commutativity for } + \\ b_1 + (b_2 + b_3) & \rightleftharpoons & (b_1 + b_2) + b_3 & \textit{associativity for } + \end{array}$$

$$\begin{array}{rclr} 1 \times b & \rightleftharpoons & b & \textit{identity for } \times \\ b_1 \times b_2 & \rightleftharpoons & b_2 \times b_1 & \textit{commutativity for } \times \\ b_1 \times (b_2 \times b_3) & \rightleftharpoons & (b_1 \times b_2) \times b_3 & \textit{associativity for } \times \end{array}$$

$$\begin{array}{rclr} 0 \times b & \rightleftharpoons & 0 & \textit{distribute over } 0 \\ (b_1 + b_2) \times b_3 & \rightleftharpoons & (b_1 \times b_3) + (b_2 \times b_3) & \textit{distribute over } + \end{array}$$

Type Isomorphisms for Finite types

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

$$\begin{array}{rclr} 0+b & \rightleftharpoons & b & \textit{identity for } + \\ b_1+b_2 & \rightleftharpoons & b_2+b_1 & \textit{commutativity for } + \\ b_1+(b_2+b_3) & \rightleftharpoons & (b_1+b_2)+b_3 & \textit{associativity for } + \end{array}$$

$$\begin{array}{rclr} 1 \times b & \rightleftharpoons & b & \textit{identity for } \times \\ b_1 \times b_2 & \rightleftharpoons & b_2 \times b_1 & \textit{commutativity for } \times \\ b_1 \times (b_2 \times b_3) & \rightleftharpoons & (b_1 \times b_2) \times b_3 & \textit{associativity for } \times \end{array}$$

$$\begin{array}{rclr} 0 \times b & \rightleftharpoons & 0 & \textit{distribute over } 0 \\ (b_1+b_2) \times b_3 & \rightleftharpoons & (b_1 \times b_3)+(b_2 \times b_3) & \textit{distribute over } + \end{array}$$

$$\frac{}{b_1 \rightleftharpoons b_1} \quad \frac{b_1 \rightleftharpoons b_2}{b_2 \rightleftharpoons b_1} \quad \frac{b_1 \rightleftharpoons b_2 \quad b_2 \rightleftharpoons b_3}{b_1 \rightleftharpoons b_3}$$

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

Type Isomorphisms with Recursive Types and Trace

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

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

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

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

Witnesses for Type Isomorphisms with Trace : Π^o

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

$$
\begin{array}{rrcll}
\textit{unfold} : & \mu x.b & \rightleftharpoons & b[\mu x.b/x] & : \textit{fold} \\
\\
\textit{zeroe} : & 0 + b & \rightleftharpoons & b & : \textit{zeroi} \\
\textit{swap}^+ : & b_1 + b_2 & \rightleftharpoons & b_2 + b_1 & : \textit{swap}^+ \\
\textit{assocl}^+ : & b_1 + (b_2 + b_3) & \rightleftharpoons & (b_1 + b_2) + b_3 & : \textit{assocr}^+ \\
\\
\textit{unite} : & 1 \times b & \rightleftharpoons & b & : \textit{uniti} \\
\textit{swap}^\times : & b_1 \times b_2 & \rightleftharpoons & b_2 \times b_1 & : \textit{swap}^\times \\
\textit{assocl}^\times : & b_1 \times (b_2 \times b_3) & \rightleftharpoons & (b_1 \times b_2) \times b_3 & : \textit{assocr}^\times \\
\\
\textit{distrib}_0 : & 0 \times b & \rightleftharpoons & 0 & : \textit{factor}_0 \\
\textit{distrib} : & (b_1 + b_2) \times b_3 & \rightleftharpoons & (b_1 \times b_3) + (b_2 \times b_3) & : \textit{factor}
\end{array}
$$

$$
\frac{}{\textit{id} : b \rightleftharpoons b} \qquad \frac{c : b_1 \rightleftharpoons b_2}{\textit{sym } c : b_2 \rightleftharpoons b_1} \qquad \frac{c_1 : b_1 \rightleftharpoons b_2 \quad c_2 : b_2 \rightleftharpoons b_3}{(c_1 \fatsemi c_2) : b_1 \rightleftharpoons b_3}
$$

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

Π^o examples : Booleans, Negation

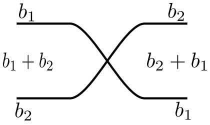
$base\ types, b = 0 \mid 1 \mid b + b \mid b \times b \mid x \mid \mu x.b$

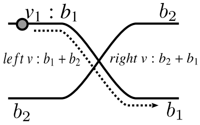
Π^o examples : Booleans, Negation

$$\textit{base types}, b = 0 \mid 1 \mid b + b \mid b \times b \mid x \mid \mu x.b$$

- We encode booleans in Π^o by $\mathit{bool} = 1 + 1$
  And thus we have
  $$\mathit{true} = \mathit{left}\ ()$$
  $$\mathit{false} = \mathit{right}\ ()$$

Π^o examples : Booleans, Negation





Thus

$$\mathit{not} : \mathit{bool} \rightleftharpoons \mathit{bool}$$
$$\mathit{not} = \mathit{swap}^+$$

and we can verify that:

$$\mathit{not}\ \mathit{true} \mapsto \mathit{false}$$
$$\mathit{not}\ \mathit{false} \mapsto \mathit{true}$$

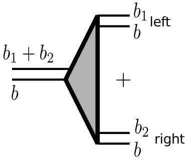
- Computation is modeled as the flow of particles in a circuit.
  - Geometry of Interaction
  - Proof Nets
  - Penrose diagrams for categories etc.

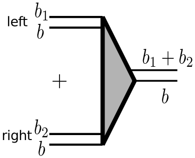
Π^o examples : Conditionals

- $\textbf{if}_c\ (\mathit{flag}, b) = \textbf{if}\ (\mathit{flag} == \mathit{true})\ \textbf{then}\ (\mathit{flag}, c(b))\ \textbf{else}\ (\mathit{flag}, b)$

Π^o examples : Conditionals

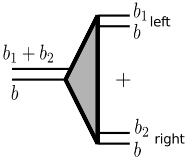
- $\textbf{if}_c\ (\mathit{flag}, b) = \textbf{if}\ (\mathit{flag} == \mathit{true})\ \textbf{then}\ (\mathit{flag}, c(b))\ \textbf{else}\ (\mathit{flag}, b)$

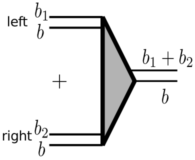


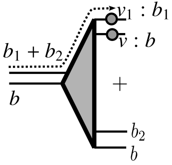


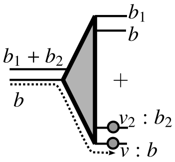
Π^o examples : Conditionals

- $\textbf{if}_c\ (flag, b) = \textbf{if}\ (flag == true)\ \textbf{then}\ (flag, c(b))\ \textbf{else}\ (flag, b)$







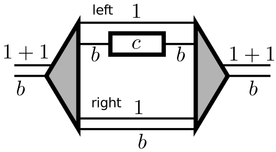


Π^o examples : Conditionals

- $\textbf{if}_c\ (\mathit{flag}, b) = \textbf{if}\ (\mathit{flag} == \mathit{true})\ \textbf{then}\ (\mathit{flag}, c(b))\ \textbf{else}\ (\mathit{flag}, b)$

Π^o examples : Conditionals

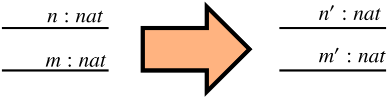
- $\mathbf{if}_c\ (\mathit{flag}, b) = \mathbf{if}\ (\mathit{flag} == \mathit{true})\ \mathbf{then}\ (\mathit{flag}, c(b))\ \mathbf{else}\ (\mathit{flag}, b)$
- $\mathbf{if}_c : \mathit{bool} \times b \rightleftharpoons \mathit{bool} \times b$
  $\mathbf{if}_c = \mathit{distrib} \fatsemi ((\mathit{id} \times c) + \mathit{id}) \fatsemi \mathit{factor}$



Π^o examples : Abstract Machines

$$\begin{aligned} \textit{Numbers}, n, m &= 0 \mid n+1 \\ \textit{Machine states} &= \langle n, n \rangle \end{aligned} \qquad \begin{aligned} \textit{Start state} &= \langle n, 0 \rangle \\ \textit{Stop State} &= \langle 0, n \rangle \end{aligned}$$

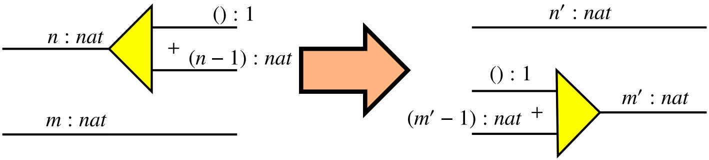
$$\langle n+1, m \rangle \mapsto \langle n, m+1 \rangle$$



Π^o examples : Abstract Machines

$$\begin{aligned} \textit{Numbers}, n, m &= 0 \mid n+1 & \qquad \textit{Start state} &= \langle n, 0 \rangle \\ \textit{Machine states} &= \langle n, n \rangle & \qquad \textit{Stop State} &= \langle 0, n \rangle \end{aligned}$$

$$\langle n+1, m \rangle \mapsto \langle n, m+1 \rangle$$

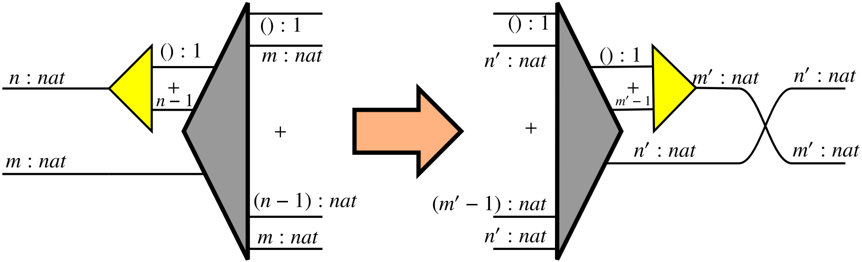


$$fold_{nat} : 1 + nat \rightleftharpoons nat : unfold_{nat}$$
$$nat = \mu x.(1 + x)$$

Π^o examples : Abstract Machines

$$\begin{array}{rcl} \textit{Numbers}, n, m & = & 0 \mid n+1 \\ \textit{Machine states} & = & \langle n, n \rangle \end{array} \qquad \begin{array}{rcl} \textit{Start state} & = & \langle n, 0 \rangle \\ \textit{Stop State} & = & \langle 0, n \rangle \end{array}$$

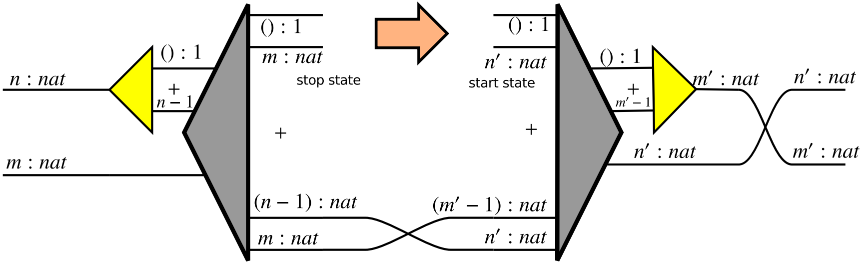
$$\langle n+1, m \rangle \mapsto \langle n, m+1 \rangle$$



Π^o examples : Abstract Machines

$$
\begin{aligned}
\textit{Numbers}, n, m &= 0 \mid n+1 & \qquad \textit{Start state} &= \langle n, 0 \rangle \\
\textit{Machine states} &= \langle n, n \rangle & \qquad \textit{Stop State} &= \langle 0, n \rangle
\end{aligned}
$$

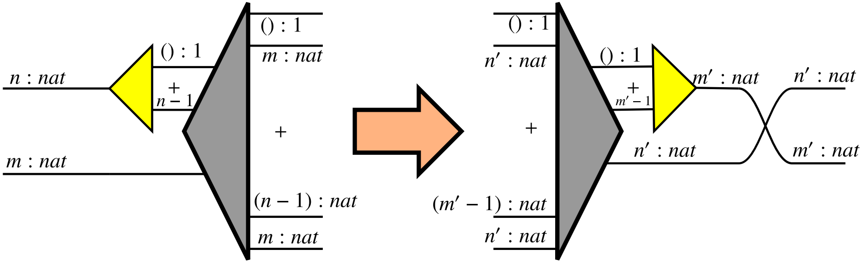
$$\langle n+1, m \rangle \mapsto \langle n, m+1 \rangle$$



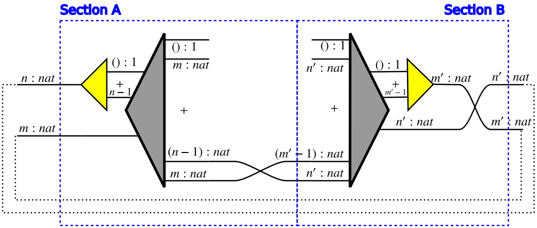
Π^o examples : Abstract Machines

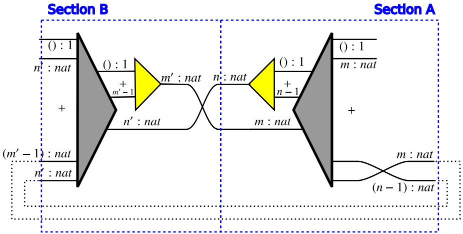
$$\begin{array}{rcl} \textit{Numbers}, n, m & = & 0 \mid n+1 \\ \textit{Machine states} & = & \langle n, n \rangle \end{array} \qquad \begin{array}{rcl} \textit{Start state} & = & \langle n, 0 \rangle \\ \textit{Stop State} & = & \langle 0, n \rangle \end{array}$$

$$\langle n+1, m \rangle \mapsto \langle n, m+1 \rangle$$



Π^o examples : Abstract Machines

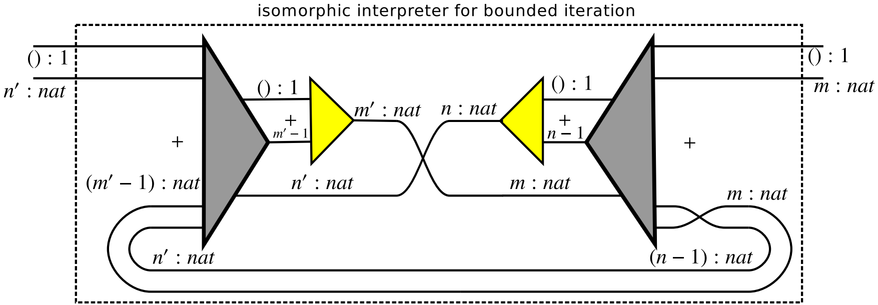




Π^o examples : Abstract Machines

$$\begin{array}{rcl} \textit{Numbers}, n, m & = & 0 \mid n+1 \\ \textit{Machine states} & = & \langle n, n \rangle \end{array} \qquad \begin{array}{rcl} \textit{Start state} & = & \langle n, 0 \rangle \\ \textit{Stop State} & = & \langle 0, n \rangle \end{array}$$

$$\langle n+1, m \rangle \mapsto \langle n, m+1 \rangle$$

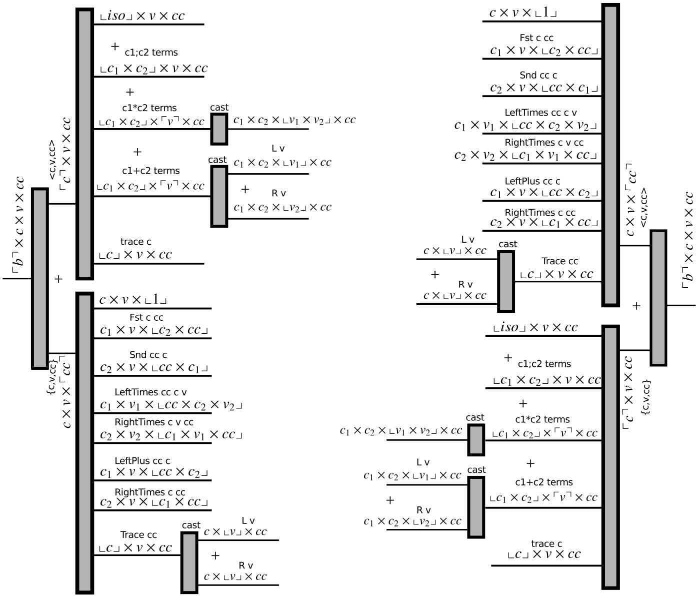


Π^o examples : Metacircular Evaluator

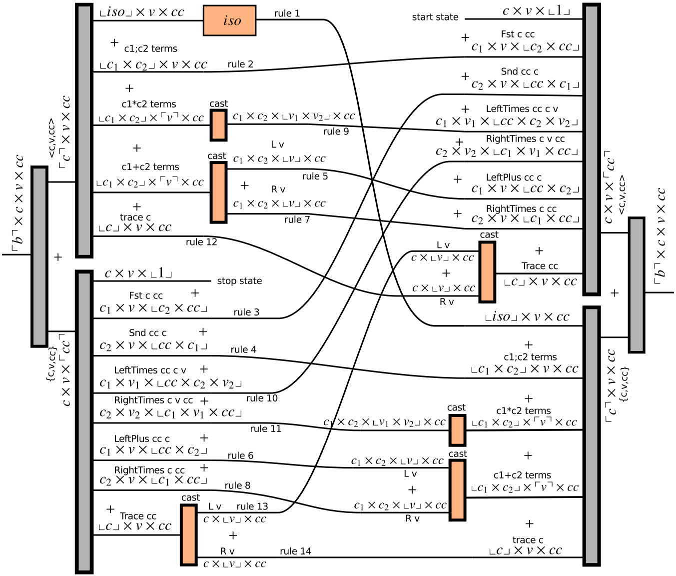
$$
\begin{array}{rcl}
\textit{Combinators}, c & = & \textit{iso} \mid c \fatsemi c \mid c \times c \mid c + c \mid \textit{trace}\ c \\
\textit{Combinator Contexts}, cc & = & \square \mid \textit{Fst}\ cc\ c \mid \textit{Snd}\ c\ cc \\
 & \mid & \textit{LeftTimes}\ cc\ c\ v \mid \textit{RightTimes}\ c\ v\ cc \\
 & \mid & \textit{LeftPlus}\ cc\ c \mid \textit{RightPlus}\ c\ cc \mid \textit{Trace}\ cc \\
\textit{Values}, v & = & () \mid (v, v) \mid L\ v \mid R\ v \\
\\
\textit{Machine states} & = & \langle c, v, cc \rangle \mid \{c, v, cc\} \\
\textit{Start state} & = & \langle c, v, \square \rangle \\
\textit{Stop State} & = & \{c, v, \square\}
\end{array}
$$

$$
\begin{array}{rcll}
\langle \textit{iso}, v, cc \rangle & \mapsto & \{\textit{iso}, \textit{iso}(v), cc\} & \textit{rule}\ 1 \\
\langle c_1 \fatsemi c_2, v, cc \rangle & \mapsto & \langle c_1, v, \textit{Fst}\ cc\ c_2 \rangle & \textit{rule}\ 2 \\
\{c_1, v, \textit{Fst}\ cc\ c_2\} & \mapsto & \langle c_2, v, \textit{Snd}\ c_1\ cc \rangle & \textit{rule}\ 3 \\
\{c_2, v, \textit{Snd}\ c_1\ cc\} & \mapsto & \{c_1 \fatsemi c_2, v, cc\} & \textit{rule}\ 4 \\
\langle c_1 + c_2, L\ v, cc \rangle & \mapsto & \langle c_1, v, \textit{LeftPlus}\ cc\ c_2 \rangle & \textit{rule}\ 5 \\
\{c_1, v, \textit{LeftPlus}\ cc\ c_2\} & \mapsto & \{c_1 + c_2, L\ v, cc\} & \textit{rule}\ 6 \\
\langle c_1 + c_2, R\ v, cc \rangle & \mapsto & \langle c_2, v, \textit{RightPlus}\ c_1\ cc \rangle & \textit{rule}\ 7 \\
\{c_2, v, \textit{RightPlus}\ c_1\ cc\} & \mapsto & \{c_1 + c_2, R\ v, cc\} & \textit{rule}\ 8 \\
\langle c_1 \times c_2, (v_1, v_2), cc \rangle & \mapsto & \langle c_1, v_1, \textit{LeftTimes}\ cc\ c_2\ v_2 \rangle & \textit{rule}\ 9 \\
\{c_1, v_1, \textit{LeftTimes}\ cc\ c_2\ v_2\} & \mapsto & \langle c_2, v_2, \textit{RightTimes}\ c_1\ v_1\ cc \rangle & \textit{rule}\ 10 \\
\{c_2, v_2, \textit{RightTimes}\ c_1\ v_1\ cc\} & \mapsto & \{c_1 \times c_2, (v_1, v_2), cc\} & \textit{rule}\ 11 \\
\langle \textit{trace}\ c, v, cc \rangle & \mapsto & \langle c, R\ v, \textit{Trace}\ cc \rangle & \textit{rule}\ 12 \\
\{c, L\ v, \textit{Trace}\ cc\} & \mapsto & \langle c, L\ v, \textit{Trace}\ cc \rangle & \textit{rule}\ 13 \\
\{c, R\ v, \textit{Trace}\ cc\} & \mapsto & \{\textit{trace}\ c, R\ v, cc\} & \textit{rule}\ 14
\end{array}
$$

Π^o examples : Metacircular Evaluator



Π^o examples : Metacircular Evaluator



$\boxed{\text{Extending } \Pi^o \text{ with Information Effects}}$

Language of Information Effects : ML_{Π^o}

Information effects are captured using an arrow-metalanguage over Π^o — similar to expressing computational effects over λ-calculus using arrows or monads.

Language of Information Effects : ML_{Π^o}

Information effects are captured using an arrow-metalanguage over Π^o — similar to expressing computational effects over λ-calculus using arrows or monads.

1. ML_{Π^o} extends Π^o with the arrow type $a : b_1 \rightharpoonup b_2$.

Language of Information Effects : ML_{Π^o}

Information effects are captured using an arrow-metalanguage over Π^o — similar to expressing computational effects over λ-calculus using arrows or monads.

1. ML_{Π^o} extends Π^o with the arrow type $a : b_1 \rightharpoonup b_2$.
2. Lifts traces, sequencing and parallel composition to the new arrow type $b_1 \rightharpoonup b_2$.
3. And adds two information effects representing the creation and erasure of information.
   - *create* : $1 \rightharpoonup b$
   - *erase* : $b \rightharpoonup 1$
4. Given the ability to *create* constants, we can construct a *clone* operator.

$\boxed{\text{Entropy Analysis Example : } \mathrm{ML}_{\Pi^o}}$

XOR and NAND example : entropy analysis

$xor(b_1, b_2) =$ **if** b_1 **then** (**if** b_2 **then** *false* **else** *true*) **else** b_2
$nand(b_1, b_2) =$ **if** b_1 **then** (**if** b_2 **then** *false* **else** *true*) **else** *true*

XOR and NAND example : entropy analysis

$xor(b_1, b_2) = \textbf{if } b_1 \textbf{ then } (\textbf{if } b_2 \textbf{ then } \mathit{false} \textbf{ else } \mathit{true}) \textbf{ else } b_2$
$nand(b_1, b_2) = \textbf{if } b_1 \textbf{ then } (\textbf{if } b_2 \textbf{ then } \mathit{false} \textbf{ else } \mathit{true}) \textbf{ else } \mathit{true}$

- Entropy of a function is $H_i - H_o$
- Entropy of input, H_i, for $bool \times bool$ is 2 bits.

XOR and NAND example : entropy analysis

$xor(b_1, b_2) = \textbf{if } b_1 \textbf{ then } (\textbf{if } b_2 \textbf{ then } \mathit{false} \textbf{ else } \mathit{true}) \textbf{ else } b_2$
$nand(b_1, b_2) = \textbf{if } b_1 \textbf{ then } (\textbf{if } b_2 \textbf{ then } \mathit{false} \textbf{ else } \mathit{true}) \textbf{ else } \mathit{true}$

- Entropy of a function is $H_i - H_o$
- Entropy of input, H_i, for $bool \times bool$ is 2 bits.
- For *xor*,
  - Probability of output being true, $P(true) = 1/2$
  - and $P(false) = 1/2$.
  - Therefore $H_o = -\sum p_i \log p_i = 1$ bits.
  - Information lost is $2 - 1 = 1$ *bit*

XOR and NAND example : entropy analysis

$xor(b_1, b_2) = \textbf{if } b_1 \textbf{ then } (\textbf{if } b_2 \textbf{ then } \mathit{false} \textbf{ else } \mathit{true}) \textbf{ else } b_2$
$nand(b_1, b_2) = \textbf{if } b_1 \textbf{ then } (\textbf{if } b_2 \textbf{ then } \mathit{false} \textbf{ else } \mathit{true}) \textbf{ else } \mathit{true}$

- Entropy of a function is $H_i - H_o$
- Entropy of input, H_i, for $bool \times bool$ is 2 bits.
- For *xor*,
  - Probability of output being true, $P(true) = 1/2$
  - and $P(false) = 1/2$.
  - Therefore $H_o = -\sum p_i \log p_i = 1$ bits.
  - Information lost is $2 - 1 = 1$ *bit*
- For *nand*,
  - Probability of output being true, $P(true) = 3/4$
  - and $P(false) = 1/4$.
  - Therefore $H_o = -\sum p_i \log p_i = 1/4 \log 4 + 3/4(\log 4 - \log 3) = 0.8$ bits.
  - Information lost is $2 - 0.8 = 1.2$ *bits*

XOR and NAND example : ML_{Π^o} implementation

- The most optimal implementation of *xor* must erase at least 1 bit i.e. there must be at least one $erase_{bool}$.

$$
\begin{aligned}
&xor : bool \times bool \rightharpoonup bool \\
&xor = distrib \ggg (not \oplus id) \\
&\quad \ggg factor \ggg (erase_{bool} \otimes id) \ggg arr\,unite
\end{aligned}
$$

- The most optimal implementation of *nand* must erase at least 1.2 bits i.e. at least two *bools* must be erased.

$$
\begin{aligned}
&nand : bool \times bool \rightharpoonup bool \\
&nand = distrib \ggg (not \oplus (erase_{bool} \ggg create_{true})) \\
&\quad \ggg factor \ggg (erase_{bool} \otimes id) \ggg arr\,unite
\end{aligned}
$$

- These minimums are captured in the structure of the program.

$$\boxed{\mathrm{LET}^o \Longrightarrow \mathrm{ML}_{\Pi^o} \Longrightarrow \Pi^o}$$

Translation 1: $\text{LET}^o \Longrightarrow \text{ML}_{\Pi^o}$

We show the translation from a first-order $\lambda^{\rightarrow}$, with sum and product types, and extended with iteration to ML_{Π^o}.

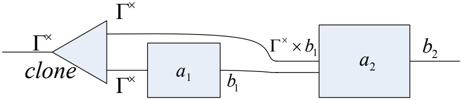
$$\Gamma \vdash e : b \Longrightarrow a : \Gamma^{\times} \rightharpoonup b$$

- The translation exposes the implicit irreversibility of the λ-calculus by requiring explicit *create* and *erase* operations.

Translation 1 : overview

- Case *let* $x = e_1$ *in* e_2:

$$\frac{\Gamma \vdash e_1 : b_1 \Longrightarrow a_1 : \Gamma^{\times} \rightharpoonup b_1 \qquad \Gamma, x : b_1 \vdash e_2 : b_2 \Longrightarrow a_2 : \Gamma^{\times} \times b_1 \rightharpoonup b_2}{\Gamma \vdash let\ x = e_1\ in\ e_2 : b_2 \Longrightarrow a : \Gamma^{\times} \rightharpoonup b_2}$$



Translation 1 : overview

- The main work of the translation is to
  - Make the implicit environment of LET^{o} explicit as the $\Gamma^{\times}$ input.
  - Thread the environment through the computation.
  - Clone/introduce constants as required.
  - Erase unwanted values.
- Much of the complexity lies in the handling of sums and *case*.
- The product fragment of this translation is much like the *CCC* semantics for $\lambda^{\rightarrow}$.

Translation 2: $\mathrm{ML}_{\Pi^o} \Longrightarrow \Pi^o$

ML_{Π^o} can be embedded in Π^o.

$$a : \Gamma^{\times} \rightharpoonup b \Longrightarrow c : h \times \Gamma^{\times} \rightleftharpoons g \times b$$

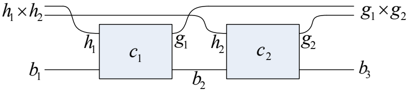
- Information effects manifest as interactions with an information environment.
  - The artifacts *h* and *g*, exposed in the types, represent this information environment.

Translation 2 : overview

- $a_1 \ggg a_2$:

$$\frac{\begin{array}{c} a_1 : b_1 \rightharpoonup b_2 \Longrightarrow c_1 : h_1 \times b_1 \rightleftharpoons g_1 \times b_2 \\ a_2 : b_2 \rightharpoonup b_3 \Longrightarrow c_2 : h_2 \times b_2 \rightleftharpoons g_2 \times b_3 \end{array}}{a_1 \ggg a_2 : b_1 \rightharpoonup b_3 \Longrightarrow c : h \times b_1 \rightleftharpoons g \times b_3}$$

We choose $h = h_1 \times h_2$ and $g = g_1 \times g_2$ and we have



Translation 2 : overview

- The main work of the translation is to shuffle h and g values through the computation.
- In the simplest sense, *create* exposes the input heap.

  $$\frac{}{\mathit{create} : 1 \rightharpoonup b \Longrightarrow c : h \times 1 \rightleftharpoons g \times b}$$

  We choose $h = b$ and $g = 1$ and we have $c = \mathit{swap}^{\times}$.
- The operator *erase* does the dual and is also realized by $\mathit{swap}^{\times}$.

Note: The paper refines this further and gives two different treatments of *create*, one for Π^{o} and one for its strong-normalizing fragment Π.

Applications and Connections

- Could serve as better basis of computation than λ-calculus for applications where information manipulation is computationally significant.
  - Quantitative information-flow security. Sabelfeld and Myers (2003)
  - Differential privacy. Dwork (2006)
  - Energy-aware computing. Ma et al. (2008); Zeng et al. (2002)
  - VLSI design. Macii and Poncino (1996)
  - Biochemical models of computation. Cardelli and Zavattaro (2008)
- Hiding in the structure of Π^o is a Dagger Symmetric Traced Monoidal Category.
  - GoI. Girard (1989)
  - Duality of Computation. Filinski (1989), Curien and Herbelin (2000)
  - Quantum Computing.
  - Categorical constructions such as $\mathcal{G}$ and **Int**.

$\boxed{\text{The Dualities of Computation}}$

Extensions: Duality

- How do we express functions in Π^o?
- How do we express continuations in Π^o?

Extensions: Duality

Not one duality, but two: negative and fractional types.

$$\begin{array}{lll} \eta^{+} & : 0 \leftrightarrow (-b) + b : & \epsilon^{+} \\ \eta^{\times} & : 1 \leftrightarrow (1/b) \times b : & \epsilon^{\times} \end{array}$$

$$\frac{\vdash v : b}{\vdash -v : -b} \qquad \frac{\vdash v : b}{\vdash 1/v : 1/b}$$

Extensions: $\Pi^{\eta\epsilon}$

Backward Information Flow.

Creation and Annihilation of entangled particles.

In many ways, we are only begining to understand these structures.

Extras

The language Π^o

Syntax of Π^o

$$
\begin{array}{rcl}
\textit{base types}, b & ::= & 0 \mid 1 \mid b + b \mid b \times b \mid x \mid \mu x.b \\
\textit{combinator types}, t & ::= & b \rightleftharpoons b \\
\textit{values}, v & ::= & () \mid \textit{left } v \mid \textit{right } v \mid (v, v) \mid \langle v \rangle \\
\\
\textit{isomorphisms}, iso & ::= & \textit{swap}^{+} \mid \textit{assocl}^{+} \mid \textit{assocr}^{+} \\
 & \mid & \textit{swap}^{\times} \mid \textit{assocl}^{\times} \mid \textit{assocr}^{\times} \mid \textit{unite} \mid \textit{uniti} \\
 & \mid & \textit{distrib} \mid \textit{factor} \mid \textit{id} \mid \textit{fold} \mid \textit{unfold} \\
\textit{combinators}, c & ::= & iso \mid \textit{sym } c \mid c \,;\, c \mid c \times c \mid c + c \mid \textit{trace } c
\end{array}
$$

- Π is the fragment of Π^o sans recursive types and trace.
  - i.e. without the red parts.

Logically reversible functions are “information preserving”

1. Logical reversibility: A function $f : b_1 \rightarrow b_2$ is logically reversible if there exists an inverse function $g : b_2 \rightarrow b_1$ such that for all values $v_1 \in b_1$ and $v_2 \in b_2$, we have: $f(v_1) = v_2$ iff $g(v_2) = v_1$. (Zuliani (2001))
2. Entropy of a variable : Let ‘b’ be a (not necessarily finite) type whose values are labeled $b^1, b^2, \ldots$. Let ξ be a random variable of type b that is equal to b^i with probability p_i. The entropy of ξ is defined as $-\sum p_i \log p_i$.
3. Output entropy of a function: Consider a function $f : b_1 \rightarrow b_2$ where b_2 is a (not necessarily finite) type whose values are labeled $b_2^1, b_2^2, \ldots$. The output entropy of the function is given by $-\sum q_j \log q_j$ where q_j indicates the probability of the output of the function to have value $b_2{}^j$.
4. Information Preservation: We say a function is *information-preserving* if its output entropy is equal to the entropy of its input.
5. Non-termination is not an observable.

We extend Toffoli's result in several ways

- Richer types, not truth tables. (Full sum and product types)
- Term language for reversible and irreversible computation.
- Type directed translation.
- Deal with infinite functions (ex. over nats).

References

Luca Cardelli and Gianluigi Zavattaro. On the computational power of biochemistry. In *Third International Conference on Algebraic Biology*, 2008.

Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *ICFP*, pages 233–243, New York, NY, USA, 2000. ACM.

Cynthia Dwork. Differential privacy. In *ICALP (2)'06*, pages 1–12, 2006.

Andrzej Filinski. Declarative continuations: an investigation of duality in programming language semantics. In *Category Theory and Computer Science*, pages 224–249, London, UK, 1989. Springer-Verlag.

J.Y. Girard. Geometry of interaction 1: Interpretation of system f. *Studies in Logic and the Foundations of Mathematics*, 127:221–260, 1989.

Xiaojun Ma, Jing Huang, and Fabrizio Lombardi. A model for computing and energy dissipation of molecular QCA devices and circuits. *J. Emerg. Technol. Comput. Syst.*, 3(4):1–30, 2008.

Enrico Macii and Massimo Poncino. Exact computation of the entropy of a logic circuit. In *Proceedings of the 6th Great Lakes Symposium on VLSI*, Washington, DC, USA, 1996. IEEE Computer Society.

Andrei Sabelfeld and Andrew Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003.

Tommaso Toffoli. Reversible computing. In *Proceedings of the Colloquium on Automata, Languages and Programming*, pages 632–644. Springer-Verlag, 1980.

Heng Zeng, Carla S. Ellis, Alvin R. Lebeck, and Amin Vahdat. Ecosystem: managing energy as a first class operating system resource. *SIGPLAN Not.*, 37(10):123–132, 2002.

P. Zuliani. Logical reversibility. *IBM J. Res. Dev.*, 45:807–818, November 2001.