Type system for automated generation of reversible circuits

Benoît Valiron, PPS, Paris Diderot, France

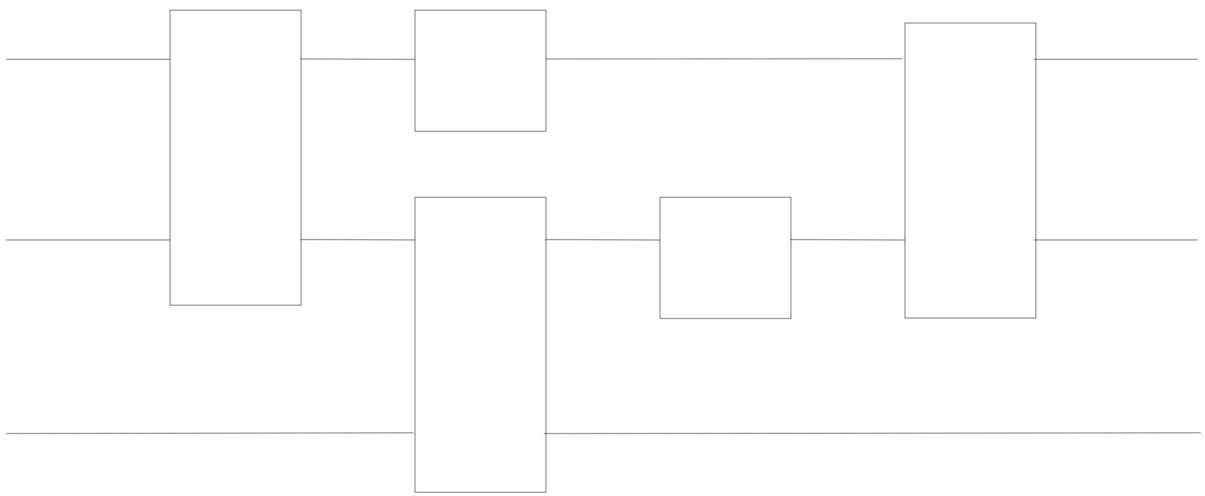
TYPES 2014

Plan of the talk

- Reversible circuits.
- Compiling a PCF-like language into reversible circuits.
- The proposed type system.
- Future Work.

Reversible circuits

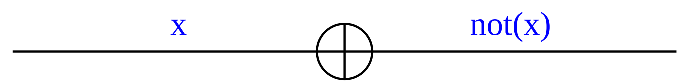
- Booleans flow on wires from left to right;
- gates modifies the booleans as they moved through;
- gates are "reversible": to reverse, have the time flow backward;
- no loops, no conditional escape.



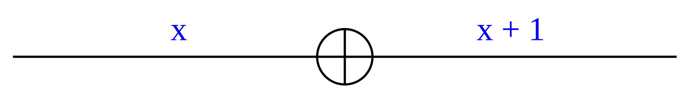
- Very useful as quantum oracles.

The building blocks

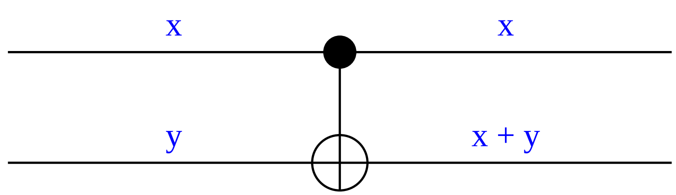
The building blocks



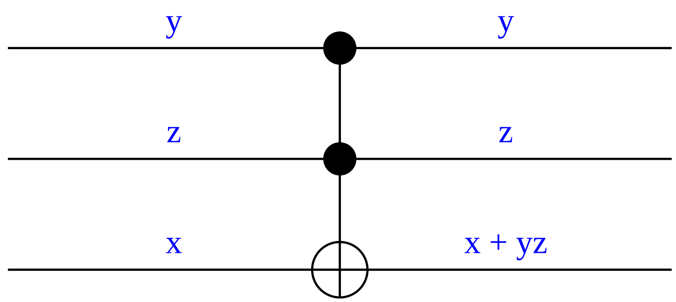
The building blocks



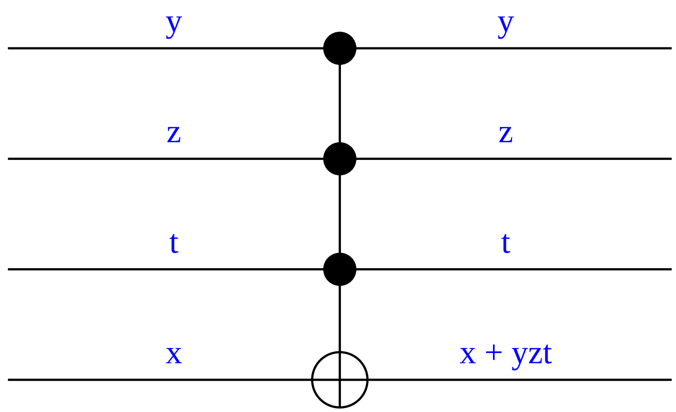
The building blocks



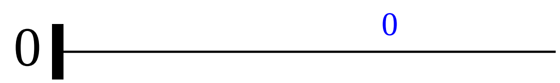
The building blocks



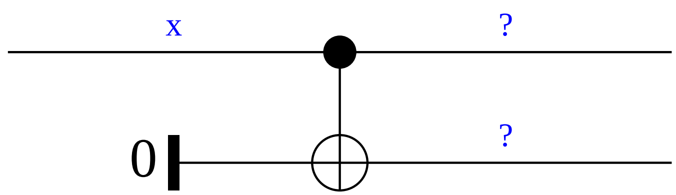
The building blocks



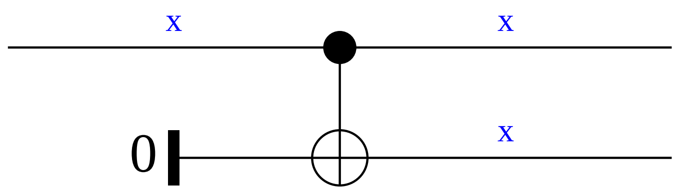
The building blocks



They can be combined



They can be combined



The problem

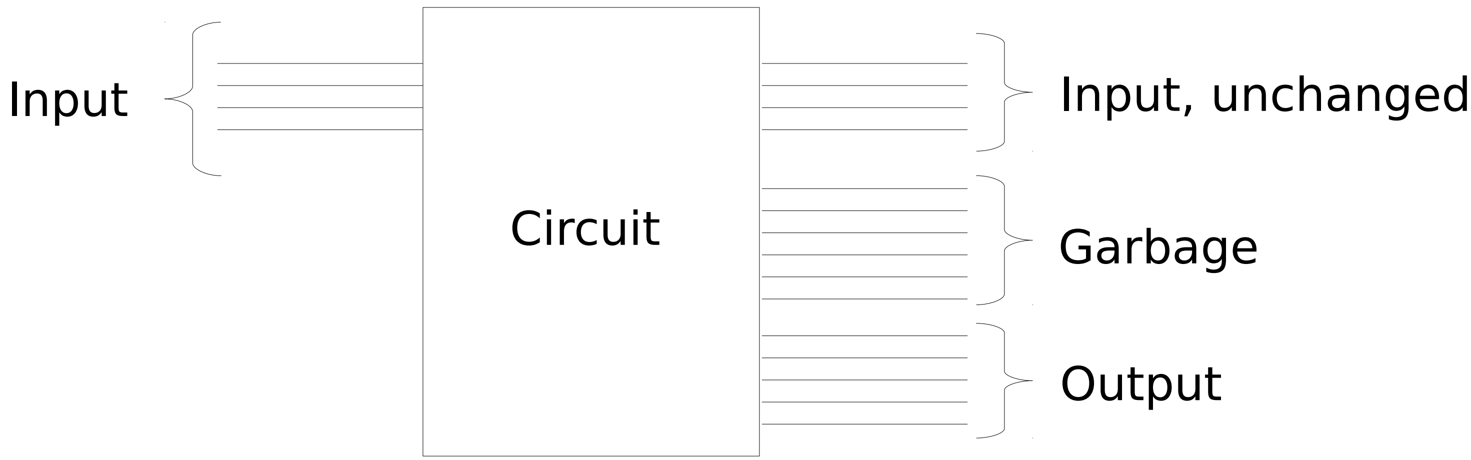
A circuit is a linear list of gates (no loop!).

Given a function $\{0,1\}^n \to \{0,1\}^m$, can you find a circuit that "compute" the function?

Hopefully as automatically as possible as quantum oracles can be quite large and complex.

A compositional approach

Landauer embeddings: circuits of the form



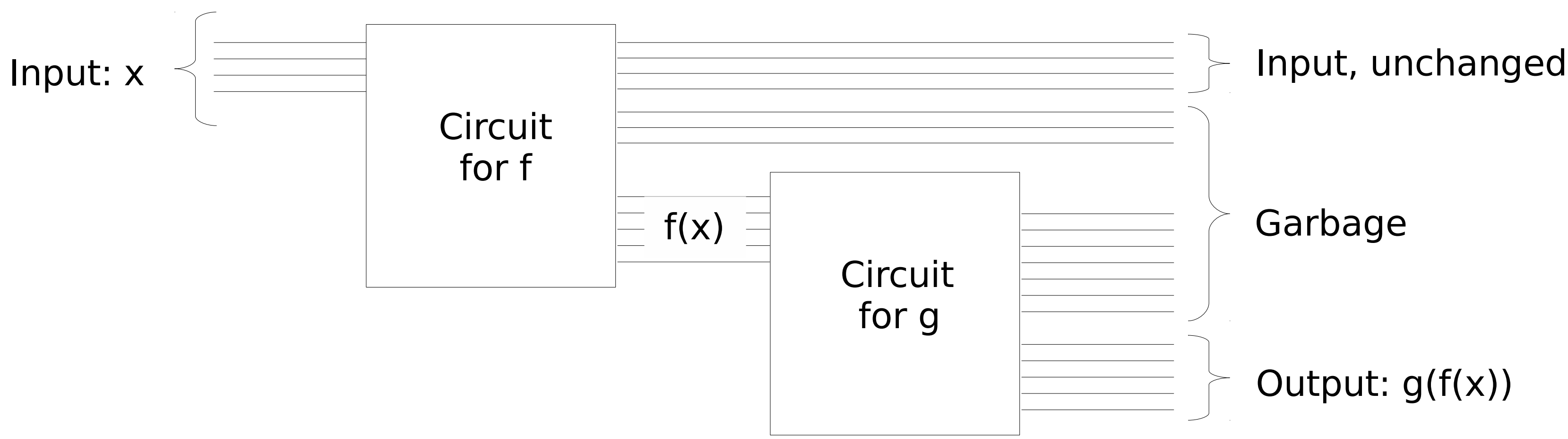
computing $\quad f : \text{Input} \longrightarrow \text{Output}.$

A compositional approach

Landauer embeddings can be composed

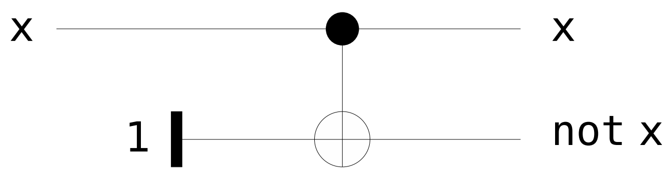
$$x \xrightarrow{f} f(x) \xrightarrow{g} g(f(x))$$

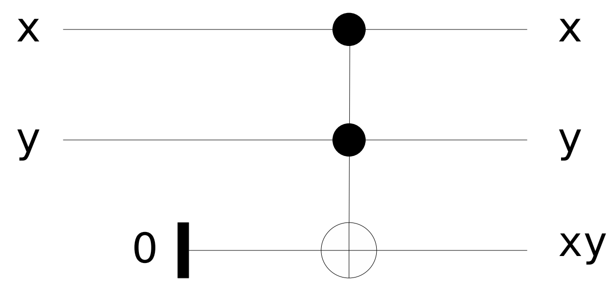
corresponds to



A compositional approach

Two elementary Landauer embeddings for `not` and `and`:



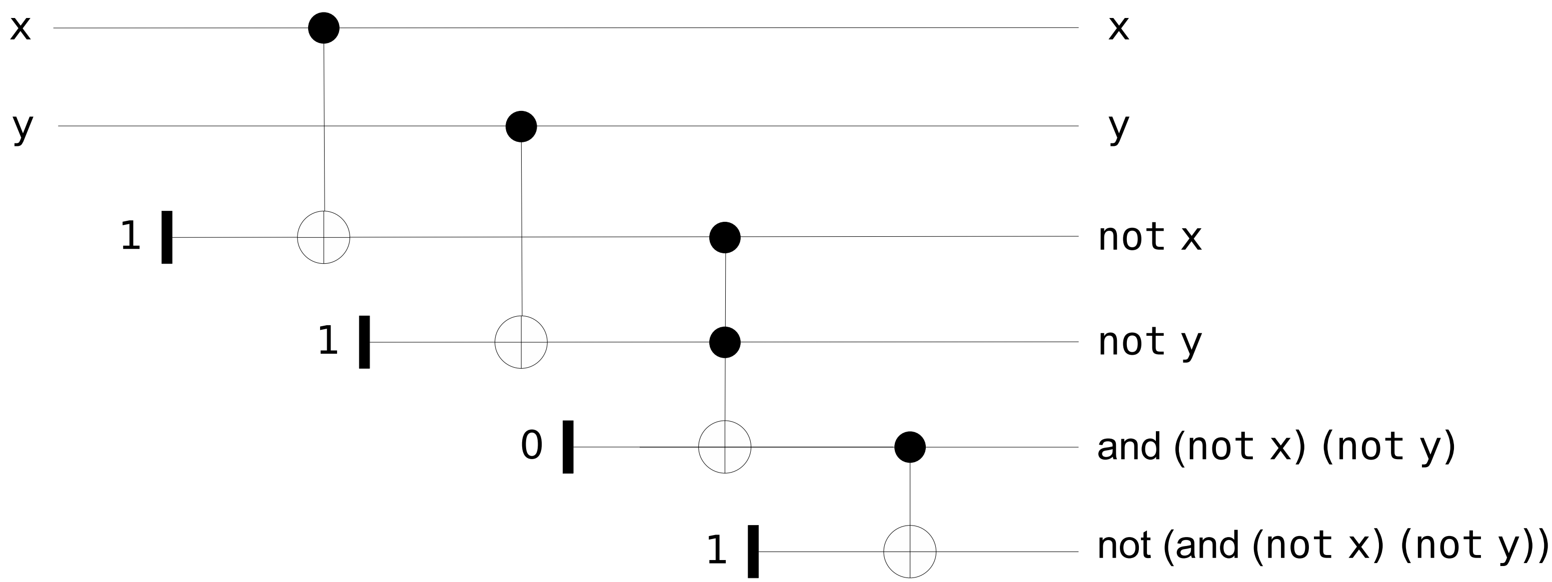


A compositional approach

Example: the disjunction

$$(x, y) \longmapsto \texttt{not}\,(\texttt{and}\,(\texttt{not}\,x)\,(\texttt{not}\,y))$$

gives the circuit



Circuits as operational semantics

Consider a lambda-calculus

$$M, N ::= x \mid \lambda x.M \mid MN \mid \texttt{and} \mid \texttt{not} \mid \cdots$$
$$A, B ::= \texttt{bit} \mid A \to B$$

An abstract machine is

$$(C, L, M)$$

- C: a circuit ;
- M: a term ;
- L: a function mapping free variables (of type `bit`) of M to wires.

Circuits as operational semantics

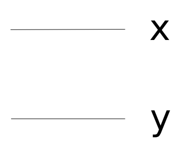
Example: the abstract machine for the term

$x, y : \texttt{bit} \vdash (((\lambda z.\lambda t.\lambda s.s\,(\texttt{and}\ t\ z))(\texttt{not}\ x))(\texttt{not}\ y))\,\texttt{not} : \texttt{bit}$

is initialized with

$M = (((\lambda z.\lambda t.\lambda s.s\,(\texttt{and}\ t\ z))(\texttt{not}\ x))(\texttt{not}\ y))\,\texttt{not}$

C is

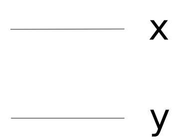


Circuits as operational semantics

Evaluating the machine in call-by-value...

$M = (((\lambda z.\lambda t.\lambda s.s\,(\texttt{and}\ t\ z))(\texttt{not}\ x))(\texttt{not}\ y))\,\texttt{not}$

C is

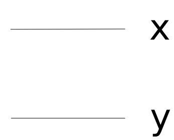


Circuits as operational semantics

Evaluating the machine in call-by-value. . .

$M = (((\lambda z.\lambda t.\lambda s.s\,(\texttt{and}\ t\ z))(\texttt{not}\ x))(\texttt{not}\ y))\,\texttt{not}$

C is

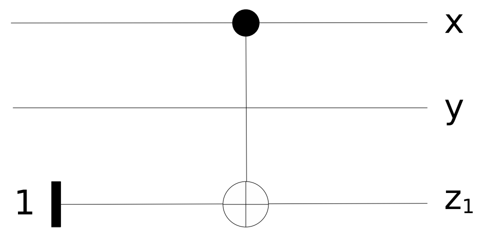


Circuits as operational semantics

Evaluating the machine in call-by-value...

$M = (((\lambda z.\lambda t.\lambda s.s\,(\texttt{and}\ t\ z))\ z_1)(\texttt{not}\ y))\,\texttt{not}$

C is

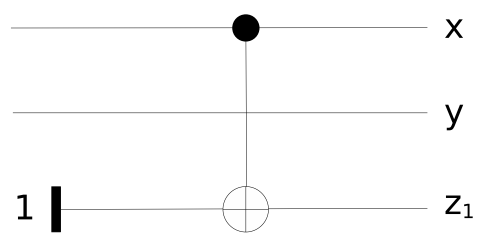


Circuits as operational semantics

Evaluating the machine in call-by-value...

$M = (((\lambda z.\lambda t.\lambda s.s\,(\texttt{and}\ t\ z))\ z_1)(\texttt{not}\ y))\,\texttt{not}$

C is

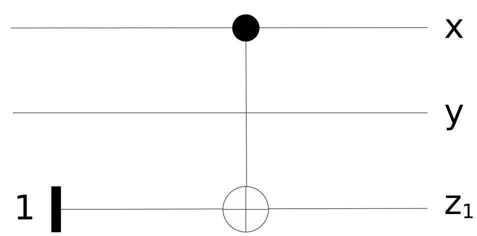


Circuits as operational semantics

Evaluating the machine in call-by-value. . .

$M = ((\lambda t.\lambda s.s\,(\texttt{and}\ t\ z_1))(\texttt{not}\ y))\,\texttt{not}$

C is

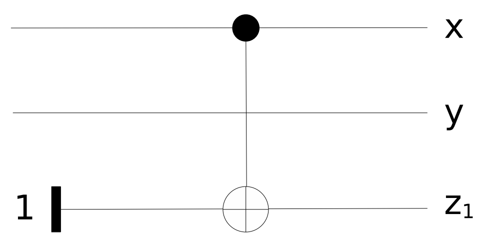


Circuits as operational semantics

Evaluating the machine in call-by-value. . .

$M = ((\lambda t.\lambda s.s\,(\texttt{and}\ t\ z_1))(\texttt{not}\ y))\,\texttt{not}$

C is

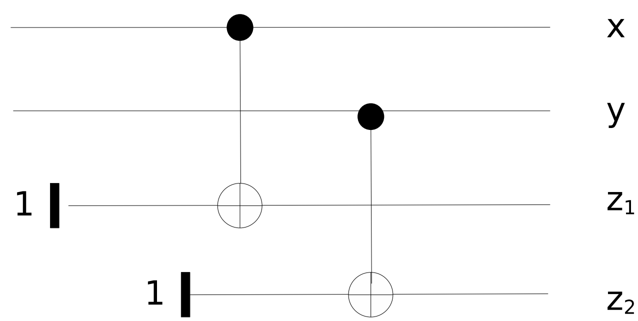


Circuits as operational semantics

Evaluating the machine in call-by-value...

$M = ((\lambda t.\lambda s.s\,(\texttt{and}\ t\ z_1))\,z_2)\,\texttt{not}$

C is

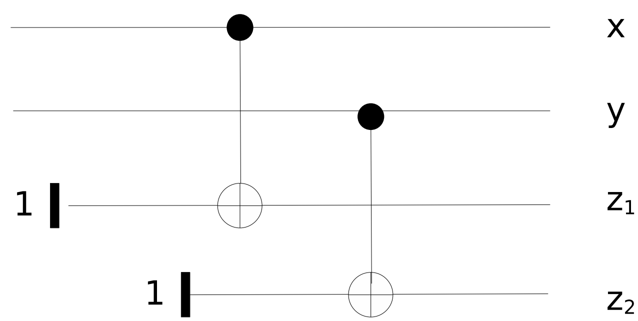


Circuits as operational semantics

Evaluating the machine in call-by-value...

$M = ((\lambda t.\lambda s.s\,(\texttt{and}\ t\ z_1))\ z_2)\,\texttt{not}$

C is

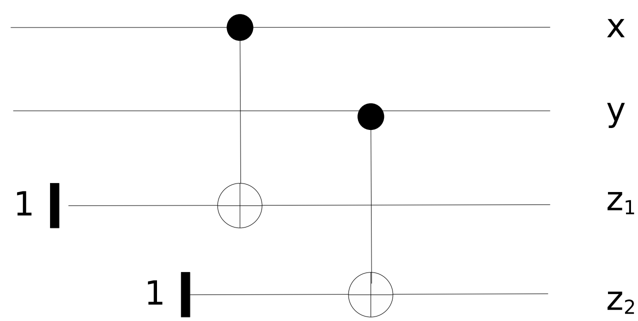


Circuits as operational semantics

Evaluating the machine in call-by-value...

$M = (\lambda s.s\,(\texttt{and}\ z_2\ z_1))\,\texttt{not}$

C is

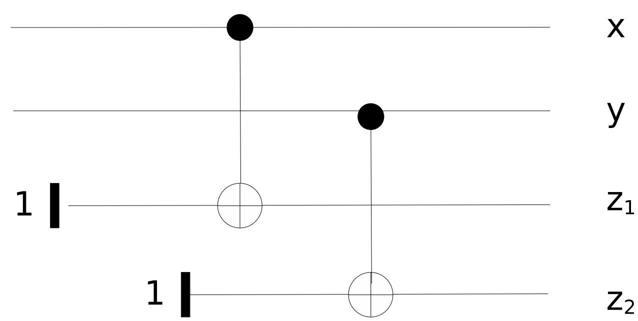


Circuits as operational semantics

Evaluating the machine in call-by-value...

$M = (\lambda s.s\,(\texttt{and}\ z_2\ z_1))\,\texttt{not}$

C is

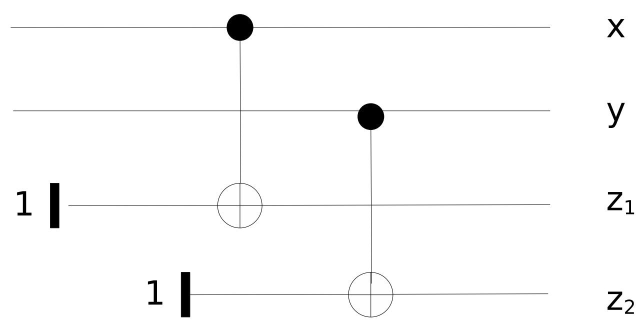


Circuits as operational semantics

Evaluating the machine in call-by-value...

$M = \texttt{not}\,(\texttt{and}\ z_2\ z_1)$

C is

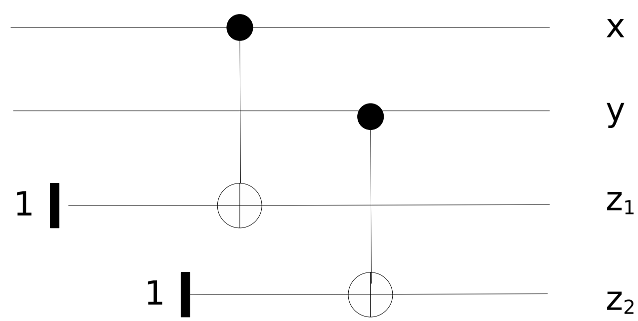


Circuits as operational semantics

Evaluating the machine in call-by-value...

$M = \texttt{not}\,(\texttt{and}\ z_2\ z_1)$

C is

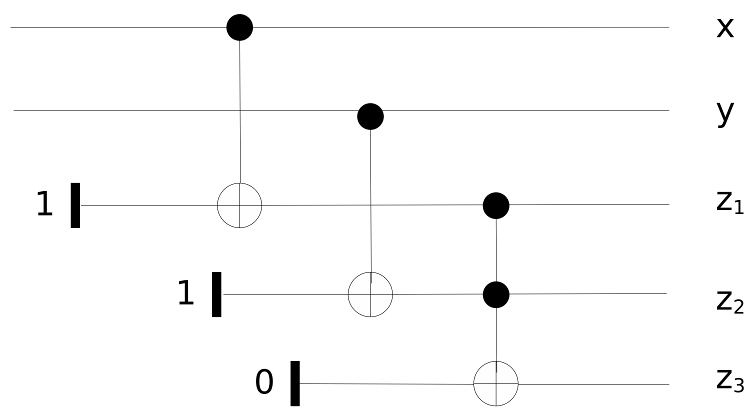


Circuits as operational semantics

Evaluating the machine in call-by-value. . .

$M = \texttt{not}\, z_3$

C is

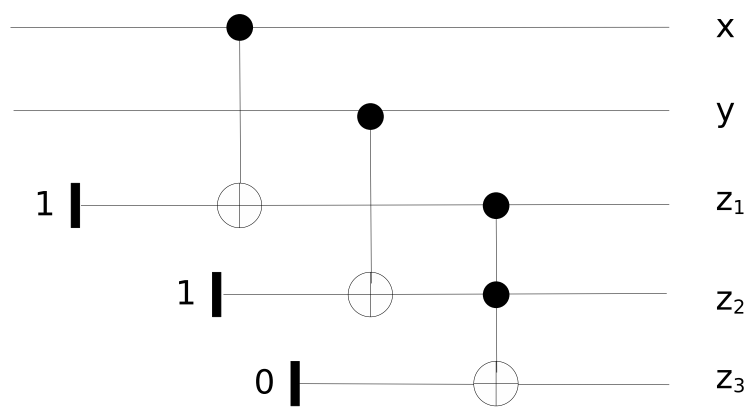


Circuits as operational semantics

Evaluating the machine in call-by-value. . .

$M = \texttt{not}\, z_3$

C is

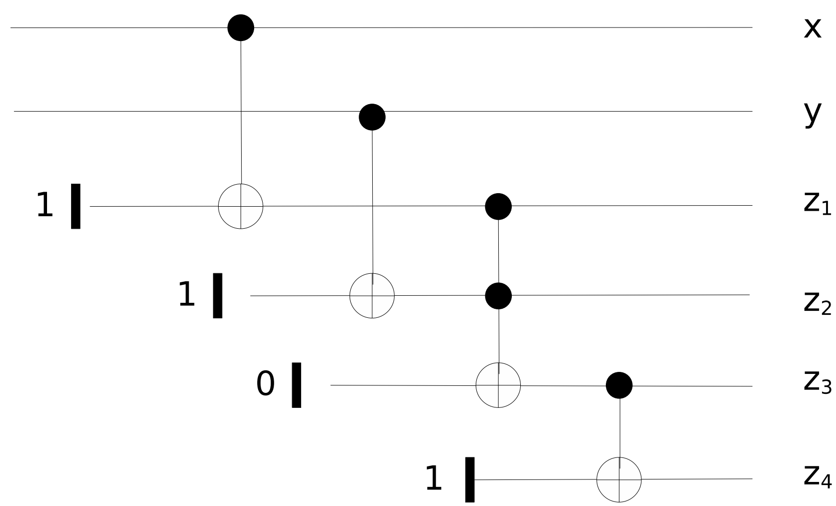


Circuits as operational semantics

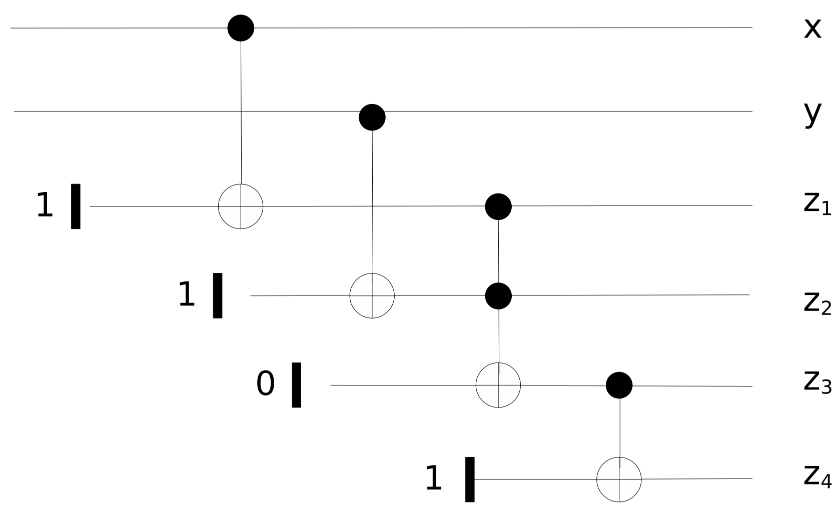
Evaluating the machine in call-by-value. . .

$M = z_4$

C is



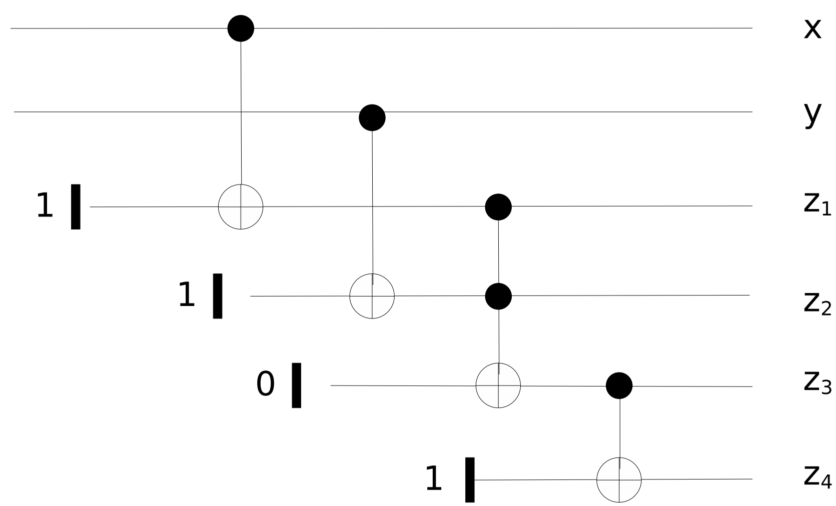
Circuits as operational semantics



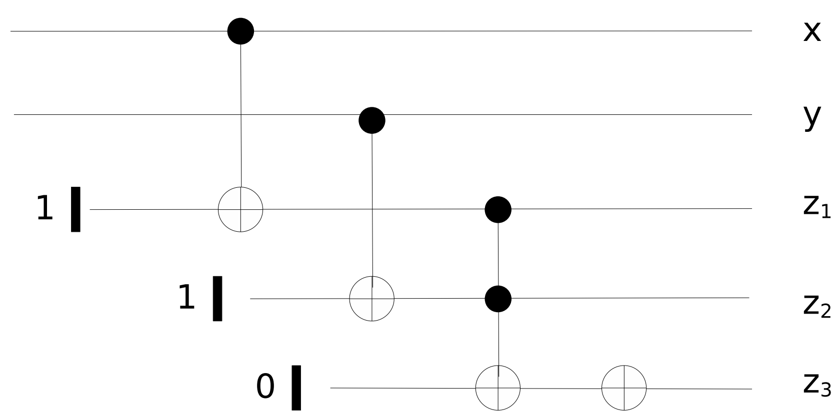
Note:

- this is the same circuit for $\texttt{not}\,(\texttt{and}\,(\texttt{not}\,x)\,(\texttt{not}\,y))$,
- the wire z_3 is not visible to the program.

Verbosity



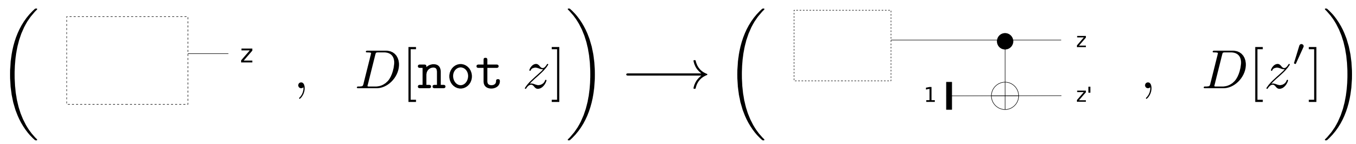
can then be replaced with



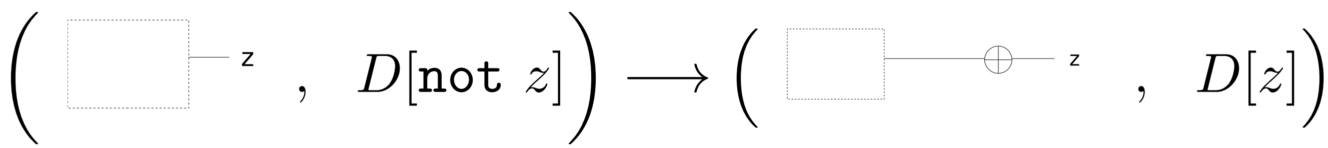
Verbosity

In general, if $D[-]$ is in evaluation position:

- If the wire z is used more than once in the program



- Else:



Easy to track for $\texttt{not}\,(\texttt{and}\,(\texttt{not}\,x)\,(\texttt{not}\,y))$,
not so much for $(((\lambda z.\lambda t.\lambda s.s\,(\texttt{and}\;t\;z))(\texttt{not}\;x))(\texttt{not}\;y))\,\texttt{not}$.

Typing wires

The problem comes from the fact that wire occupancy by a control is not monitored. Types can be used for that purpose.

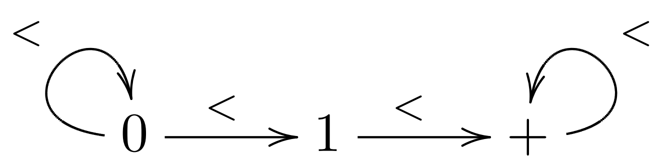
Idea: Wires are described as sorts for the type `bit`, and sorts are themselves typed with the “occupancy level” of the wire.

$$M, N ::= x \mid \lambda x.M \mid MN \mid \texttt{and} \mid \texttt{not}_\alpha \ldots$$

$$A, B ::= \alpha \mid A \to B$$

$$\tau ::= 0 \mid 1 \mid +.$$

and sorts enjoy a transitive relation:



Typing wires

$$\alpha_1 : \tau_1 \ldots \alpha_n : \tau_n \mid x_1 : A_1 \ldots x_m : A_m \vdash M : B$$

$$\frac{|\Delta| = |A|}{\Delta \mid x : A \vdash x : A} \quad \frac{\Delta \mid \Gamma, x : A \vdash M : B}{\Delta \mid \Gamma \vdash \lambda x.M : A \to B} \quad \frac{\begin{array}{c}\Gamma_1 \mid \Delta \vdash N : A \\ \Gamma_2 \mid \Delta \vdash M : A \to B\end{array}}{\Gamma_1 \cup \Gamma_2 \mid \Delta \vdash MN : B}$$

(plus weakening) where

$$\begin{array}{rl} & (\alpha_1 : \tau_1 \ldots \alpha_n : \tau_n, \quad \beta_1 : \sigma_1 \ldots \beta_m : \sigma_m) \quad \cup \\ & (\alpha_1 : \tau_1' \ldots \alpha_n : \tau_n', \quad \beta_{m+1} : \sigma_{m+1} \ldots \beta_k : \sigma_k) \\ = & (\alpha_1 : \max(\tau_1, \tau_1') \ldots \alpha_n : \max(\tau_n, \tau_n'), \quad \beta_1 : \sigma_1 \ldots \beta_k : \sigma_k) \end{array}$$

with $\max(\tau, \tau') = \min(\sigma \mid \tau, \tau' \leq \sigma,\ \sigma > \tau \text{ or } \sigma > \tau')$.

Typing wires

Example:

$$\frac{\alpha : 0 \mid x : \alpha \vdash M : A \to B \quad \alpha : 0 \mid x : \alpha \vdash N : A}{\alpha : \max(0,0) \mid x : \alpha \vdash MN : B}$$

Typing wires

Example:

$$\frac{\alpha : 0 \mid x : \alpha \vdash M : A \to B \quad \alpha : 0 \mid x : \alpha \vdash N : A}{\alpha : \quad 0 \quad \mid x : \alpha \vdash MN : B}$$

Since $0 \leq 0$ and $0 > 0$.

That is, the wire α is not used in the circuit generated by MN.

Typing wires

Example:

$$\frac{\alpha : 1 \mid x : \alpha \vdash M : A \to B \quad \alpha : 0 \mid x : \alpha \vdash N : A}{\alpha : \max(1, 0) \mid x : \alpha \vdash MN : B}$$

Typing wires

Example:

$$\frac{\alpha : 1 \mid x : \alpha \vdash M : A \to B \quad \alpha : 0 \mid x : \alpha \vdash N : A}{\alpha : \quad 1 \quad \mid x : \alpha \vdash MN : B}$$

Since $0, 1 \leq 1$ and $1 > 0$.

That is, the wire α is used only once in the circuit generated by MN.

Typing wires

Example:

$$\frac{\alpha : 1 \mid x : \alpha \vdash M : A \to B \quad \alpha : 1 \mid x : \alpha \vdash N : A}{\alpha : \max(1,1) \mid x : \alpha \vdash MN : B}$$

Typing wires

Example:

$$\frac{\alpha : 1 \mid x : \alpha \vdash M : A \to B \quad \alpha : 1 \mid x : \alpha \vdash N : A}{\alpha : \qquad + \qquad \mid x : \alpha \vdash MN : B}$$

Since $1 \leq +$, $1 \not< 1$ and $+ > 1$.

The wire α is used more than once in the circuit generated by MN.

Types and constant terms

$$\frac{\tau_1 \geq 1}{\alpha : \tau_1, \beta : \tau_2 \mid \emptyset \vdash \mathtt{not}_\alpha : \alpha \to \beta}$$

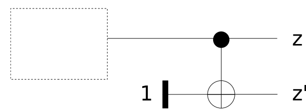
$$\frac{\tau_1, \tau_2 \geq 1}{\alpha : \tau_1, \beta : \tau_2, \gamma : \tau_3 \mid \emptyset \vdash \mathtt{and} : \alpha \to \beta \to \gamma}$$

Using types in the reduction

Suppose that $\Delta, \beta : \tau \mid \Gamma \vdash M : \alpha$, and that $M = D[\texttt{not}_\beta\ z]$ is in evaluation position.

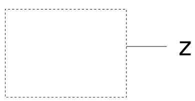
- If $\tau = +$, then

(

, $D[\texttt{not}_\beta\ z]$) $\longrightarrow$ (

, $D[z']$)

- Else:

(

, $D[\texttt{not}_\beta\ z]$) $\longrightarrow$ (

, $D[z]$)

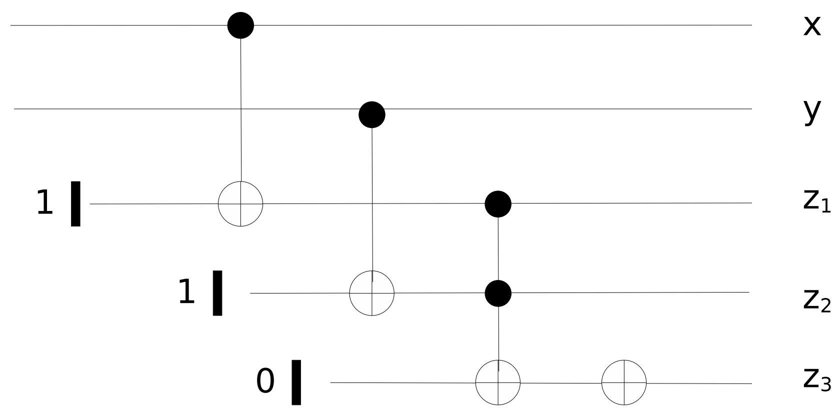
Revisiting the example

Both of the terms $\texttt{not}_\epsilon\,(\texttt{and}\,(\texttt{not}_\alpha\,x)\,(\texttt{not}_\beta\,y))$
and $(((\lambda z.\lambda t.\lambda s.s\,(\texttt{and}\ t\ z))(\texttt{not}_\alpha\ x))(\texttt{not}_\beta\ y))\,\texttt{not}_\epsilon$

can be typed with the context

$$\ldots, \alpha : +, \beta : +, \epsilon : 1, \gamma : 0 \mid x : \alpha, y : \beta \vdash - : \gamma$$

and the circuit is



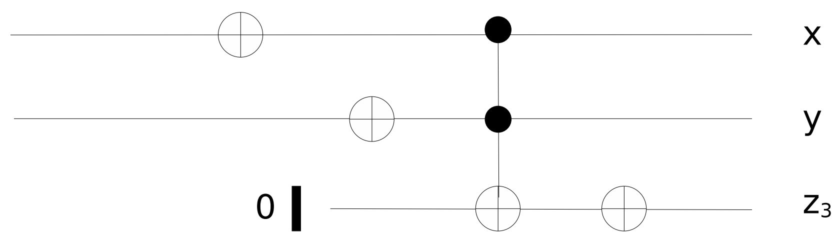
Revisiting the example

Both of the terms $\mathtt{not}_\epsilon\,(\mathtt{and}\,(\mathtt{not}_\alpha\,x)\,(\mathtt{not}_\beta\,y))$
and $(((\lambda z.\lambda t.\lambda s.s\,(\mathtt{and}\ t\ z))(\mathtt{not}_\alpha\ x))(\mathtt{not}_\beta\ y))\,\mathtt{not}_\epsilon$

can also be typed with the context

$$\ldots, \alpha : 1, \beta : 1, \epsilon : 1, \gamma : 0 \mid x : \alpha, y : \beta \vdash - : \gamma$$

and the circuit is instead



but note that we use the fact that x and y are only used once each.

The result

For a function $x_1 : \texttt{bit}, \ldots x_n : \texttt{bit} \vdash M : \texttt{bit}$, we therefore have three operational semantics:

- Regular call-by-value beta-reduction when $x_1 \ldots x_n$ are fed with concrete booleans.
- Verbose circuit-generation.
- Smart circuit-generation.

They all correspond to the same boolean function, and the verbose circuit is obviously always larger than the smart one.

Conclusion and future steps

- A step towards automation in the design of quantum oracle.
- Possible extensions:
  - parametricity in wire naming;
  - lists (e.g. of bits);
  - in general: complete PCF.
- It does not however capture all possible optimizations:
  - eta-conversion (code factorization);
  - evaluation of constants (e.g. `not true`).
- How to measure "smartness" ?