



Compiling a Quantum Programming Language

Brett Giles

`gilesb@cpsc.ucalgary.ca`

University of Calgary

Block QPL

- Defined in Dr. Selinger's paper, "Towards a Quantum Programming Language"
- Basic programming language operations.
- Two types: bit and qbit.

In the following P, Q represent statements, L lists of statements, (on the next slide), b_i, q_i, X legal identifiers, S a transform, B a block, Γ a list of type constraints and bold text keywords of the language.

- Programs ::= $B \mid \text{export proc } X : \Gamma \rightarrow \Gamma \{P\}$
- Blocks $B ::= \{L\}$
- Lists of statements $L ::= P \mid P; L$

Block QPL Statements

Statements $P, Q ::=$

new bit $b := 0$ | **new qbit** $q := 0$

| $b := 0$ | $b := 1$

| $q_1, \dots, q_n^* = S$

| **skip**

| B

| **if** b **then** P **else** Q | **measure** q **then** P **else** Q

| **while** b **do** P

| **import proc** $X : \Gamma \rightarrow \Gamma$ **in** Q

| **proc** $X : \Gamma \rightarrow \Gamma$ $\{P\}$ **in** Q

| **call** $X(x_1, \dots, x_n)$

Combining classical and quantum data in Quantum Flow Charts

- ⑥ Recall we only have two types, bit and qbit, with typing contexts.
- ⑥ Semantically, an edge labeled with n bits and m qbits can be replaced by 2^n edges each labeled with m qbits.
- ⑥ The state for the above is a 2^n -tuple (A_0, \dots, A_{2^n-1}) of density matrices each in $\mathbb{C}^{2^m \times 2^m}$
- ⑥ Extend the standard linear algebra operations on matrices via operation on the component and summing as needed.

Semantics of QPL

Define signatures as lists of non-zero natural numbers. (A signature is $\rho = n_1, \dots, n_s$.) We may associate a complex vector space

$$V_\rho = \mathbb{C}^{n_1 \times n_1} \times \dots \times \mathbb{C}^{n_s \times n_s}.$$

Then consider the category \mathbb{V} :

Objects: Signatures

Maps: $f : \rho \rightarrow \rho' \iff f$ is a complex linear function
 $f : V_\rho \rightarrow V_{\rho'}$

Identity: Identity function

Composition: Inherited

Semantics of QPL, cont'd

A *superoperator* is a linear function F that:

- ⑥ is positive. (A positive $\implies F(A)$ positive.)
- ⑥ is completely positive. ($\text{id}_\tau \otimes F$ is positive for all signatures τ)
- ⑥ $\text{trace}(F(A)) \leq \text{trace}(A)$, for all positive A .

Then the semantics of QPL are given by the subcategory \mathbb{Q} of \mathbb{V} which has the same objects and has the morphisms restricted to superoperators.

Interpretation of statements.

[[new bit $b := 0$]]

$newbit :: \mathbf{I} \rightarrow \mathbf{bit} : a \mapsto (a, 0)$

[[new qbit $q := 0$]]

$newqbit :: \mathbf{I} \rightarrow \mathbf{qbit} : a \mapsto \begin{pmatrix} a & 0 \\ 0 & 0 \end{pmatrix}$

[[discard b]]

$discardbit :: \mathbf{bit} \rightarrow \mathbf{I} : (a, b) \mapsto a + b$

[[discard q]]

$discardqbit :: \mathbf{qbit} \rightarrow \mathbf{I} : \begin{pmatrix} a & b \\ c & d \end{pmatrix} \mapsto a + d$

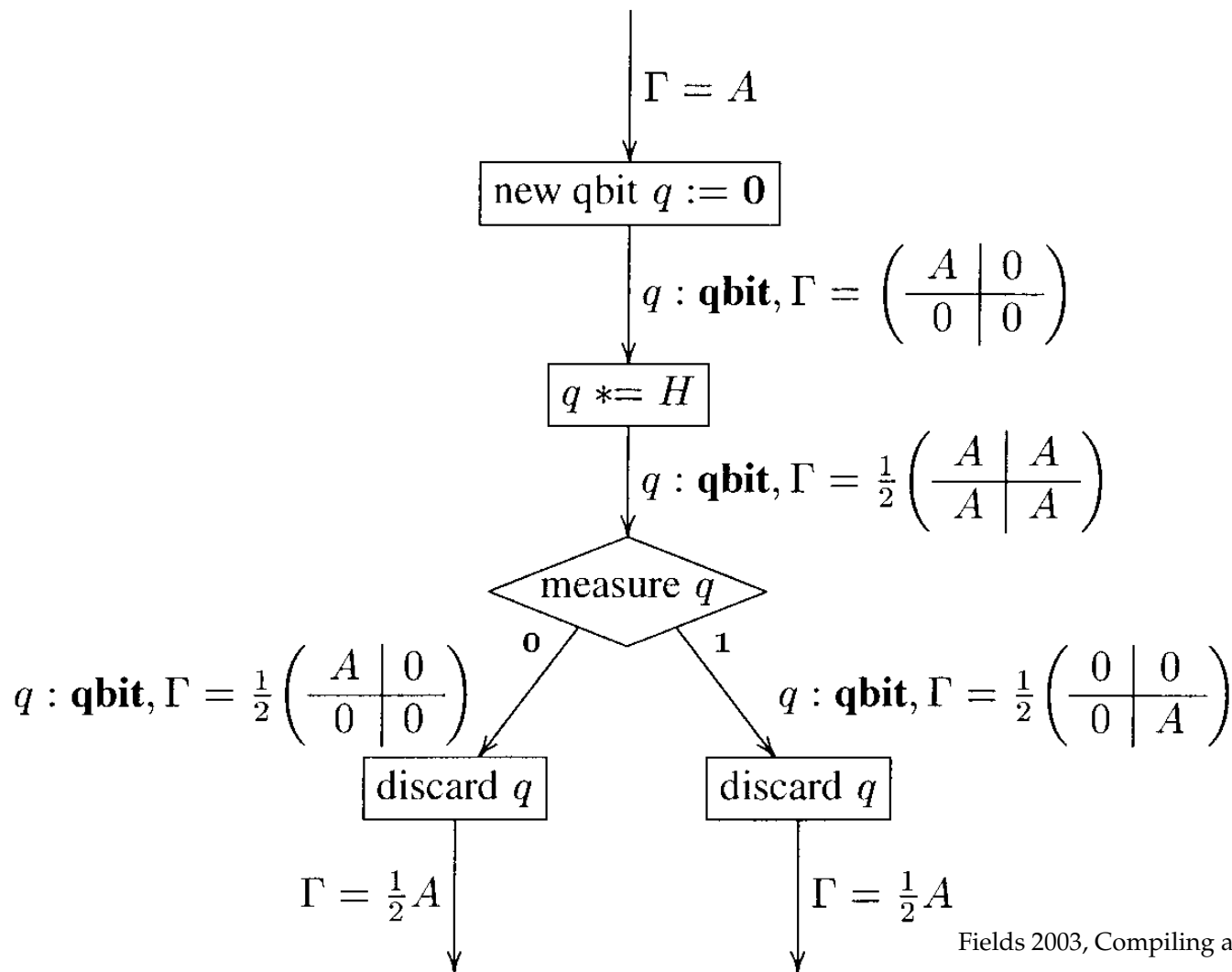
[[measure q]]

$measure :: \mathbf{qbit} \rightarrow \mathbf{qbit} \oplus \mathbf{qbit} :$

$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \mapsto \left(\begin{pmatrix} a & 0 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 0 & d \end{pmatrix} \right)$

Examples of quantum flow charts

Fair Coin Toss



Example - a quantum coinflip.

```
1  { proc cf : a:bit -> a:bit
2    {
3      new qbit q := 0;
4      q *= H;
5      measure q then
6        a := 0
7      else
8        a := 1;
9    } in
10   {
11     new bit x := 0;
12     call cf(x);
13     while x do
14       call cf(x);
15   }
16 }
```

Example - adding two bits.

```
1 export proc addwcarry:
2     r:bit, carry:bit, a:bit, b:bit
3     -> r:bit, carry:bit, a:bit, b:bit
4 {   carry:=0;
5     if a then
6         if b then {
7             r := 0;
8             carry := 1;
9         }
10        else
11            r := 1;
12    else
13        if b then
14            r:= 1;
15        else
16            r:= 0;
17 }
```

Compiling Block-QPL

Of the four standard phases of a compiler (Lex, Parse, Semantic Analysis, and Code Generation), semantic analysis was the only one with somewhat different characteristics.

This is because qbits may not be copied. For example, when doing a unitary transform on 2 qbits, we may not use the same qbit. As another example, when calling a procedure with more than one qbit, they must all be distinct.

Emulating the Quantum Machine

We considered two possible ways to do this:

- ⑥ When running a coin-flip, for example, set values according to the probabilities and then show the values of any bits or measured qbits at the end.
- ⑥ OR, directly implement the semantics, allowing one to view the probabilities of the bit values or qbit matrices along the way.

We felt the second was the most advantageous, especially when designing quantum algorithms.

A Quantum stack machine

- ⑥ A standard machine to implement simple classical language is stack based.
- ⑥ We use a tree as a “stack”, where each item in the stack is either a bit (has two branches) or qbit (has four branches).
- ⑥ Note a conceptual difference from classical stack machines is that elements are “rotated” to the top to operate on them, rather than duplicating.

Quantum Stack machine instructions

<i>newbit, discardbit, setbit, unsetbit</i>	bit operations
<i>newqbit, discardqbit, utrans(8)</i>	qbit operations
<i>if, measure, bringup, push, switch, pop, merge, initial</i>	stack manipulations

All of these are of the type:

$qStack \times InsStream \times Dump \rightarrow qStack \times InsStream \times Dump$

Example transitions during an IF



S |

$if(b); push; c_0; switch; c_1; pop; merge; c$ |

D

Where we have S is the stack.

Example transitions during an IF

S |

$if(b); push; c_0; switch; c_1; pop; merge; c$ |

D

Where we have $if(b); push \dots$ is the instruction stream.

Example transitions during an IF



S |

$if(b); push; c_0; switch; c_1; pop; merge; c$ |

D

Where we have D is the Dump where stacks may be placed.

Example transitions during an IF

$S \mid \text{if}(b); \text{push}; c_0; \text{switch}; c_1; \text{pop}; \text{merge}; c \mid D$

$\rightarrow (S_0, S_1) \mid \text{push}; c_0; \text{switch}; c_1; \text{pop}; \text{merge}; c \mid D$

Example transitions during an IF

$S \mid \text{if}(b); \text{push}; c_0; \text{switch}; c_1; \text{pop}; \text{merge}; c \mid D$

$\rightarrow (S_0, S_1) \mid \text{push}; c_0; \text{switch}; c_1; \text{pop}; \text{merge}; c \mid D$

$\rightarrow S_0 \mid c_0; \text{switch}; c_1; \text{pop}; \text{merge}; c \mid S_1 : D$

Example transitions during an IF

$S \mid \text{if}(b); \text{push}; c_0; \text{switch}; c_1; \text{pop}; \text{merge}; c \mid D$

$\rightarrow (S_0, S_1) \mid \text{push}; c_0; \text{switch}; c_1; \text{pop}; \text{merge}; c \mid D$

$\rightarrow S_0 \mid c_0; \text{switch}; c_1; \text{pop}; \text{merge}; c \mid S_1 : D$

$\dots \rightarrow S'_0 \mid \text{switch}; c_1; \text{pop}; \text{merge}; c \mid S_1 : D$

Example transitions during an IF

$S \mid \text{if}(b); \text{push}; c_0; \text{switch}; c_1; \text{pop}; \text{merge}; c \mid D$

$\rightarrow (S_0, S_1) \mid \text{push}; c_0; \text{switch}; c_1; \text{pop}; \text{merge}; c \mid D$

$\rightarrow S_0 \mid c_0; \text{switch}; c_1; \text{pop}; \text{merge}; c \mid S_1 : D$

$\dots \rightarrow S'_0 \mid \text{switch}; c_1; \text{pop}; \text{merge}; c \mid S_1 : D$

$\rightarrow S_1 \mid c_1; \text{pop}; \text{merge}; c \mid S'_0 : D$

Example transitions during an IF

$S \mid \text{if}(b); \text{push}; c_0; \text{switch}; c_1; \text{pop}; \text{merge}; c \mid D$

$\rightarrow (S_0, S_1) \mid \text{push}; c_0; \text{switch}; c_1; \text{pop}; \text{merge}; c \mid D$

$\rightarrow S_0 \mid c_0; \text{switch}; c_1; \text{pop}; \text{merge}; c \mid S_1 : D$

$\dots \rightarrow S'_0 \mid \text{switch}; c_1; \text{pop}; \text{merge}; c \mid S_1 : D$

$\rightarrow S_1 \mid c_1; \text{pop}; \text{merge}; c \mid S'_0 : D$

$\dots \rightarrow S'_1 \mid \text{pop}; \text{merge}; c \mid S'_0 : D$

Example transitions during an IF

$S \mid \text{if}(b); \text{push}; c_0; \text{switch}; c_1; \text{pop}; \text{merge}; c \mid D$

$\rightarrow (S_0, S_1) \mid \text{push}; c_0; \text{switch}; c_1; \text{pop}; \text{merge}; c \mid D$

$\rightarrow S_0 \mid c_0; \text{switch}; c_1; \text{pop}; \text{merge}; c \mid S_1 : D$

$\dots \rightarrow S'_0 \mid \text{switch}; c_1; \text{pop}; \text{merge}; c \mid S_1 : D$

$\rightarrow S_1 \mid c_1; \text{pop}; \text{merge}; c \mid S'_0 : D$

$\dots \rightarrow S'_1 \mid \text{pop}; \text{merge}; c \mid S'_0 : D$

$\rightarrow (S'_0, S'_1) \mid \text{merge}; c \mid D$

Example transitions during an IF

$S \mid \text{if}(b); \text{push}; c_0; \text{switch}; c_1; \text{pop}; \text{merge}; c \mid D$

$\rightarrow (S_0, S_1) \mid \text{push}; c_0; \text{switch}; c_1; \text{pop}; \text{merge}; c \mid D$

$\rightarrow S_0 \mid c_0; \text{switch}; c_1; \text{pop}; \text{merge}; c \mid S_1 : D$

$\dots \rightarrow S'_0 \mid \text{switch}; c_1; \text{pop}; \text{merge}; c \mid S_1 : D$

$\rightarrow S_1 \mid c_1; \text{pop}; \text{merge}; c \mid S'_0 : D$

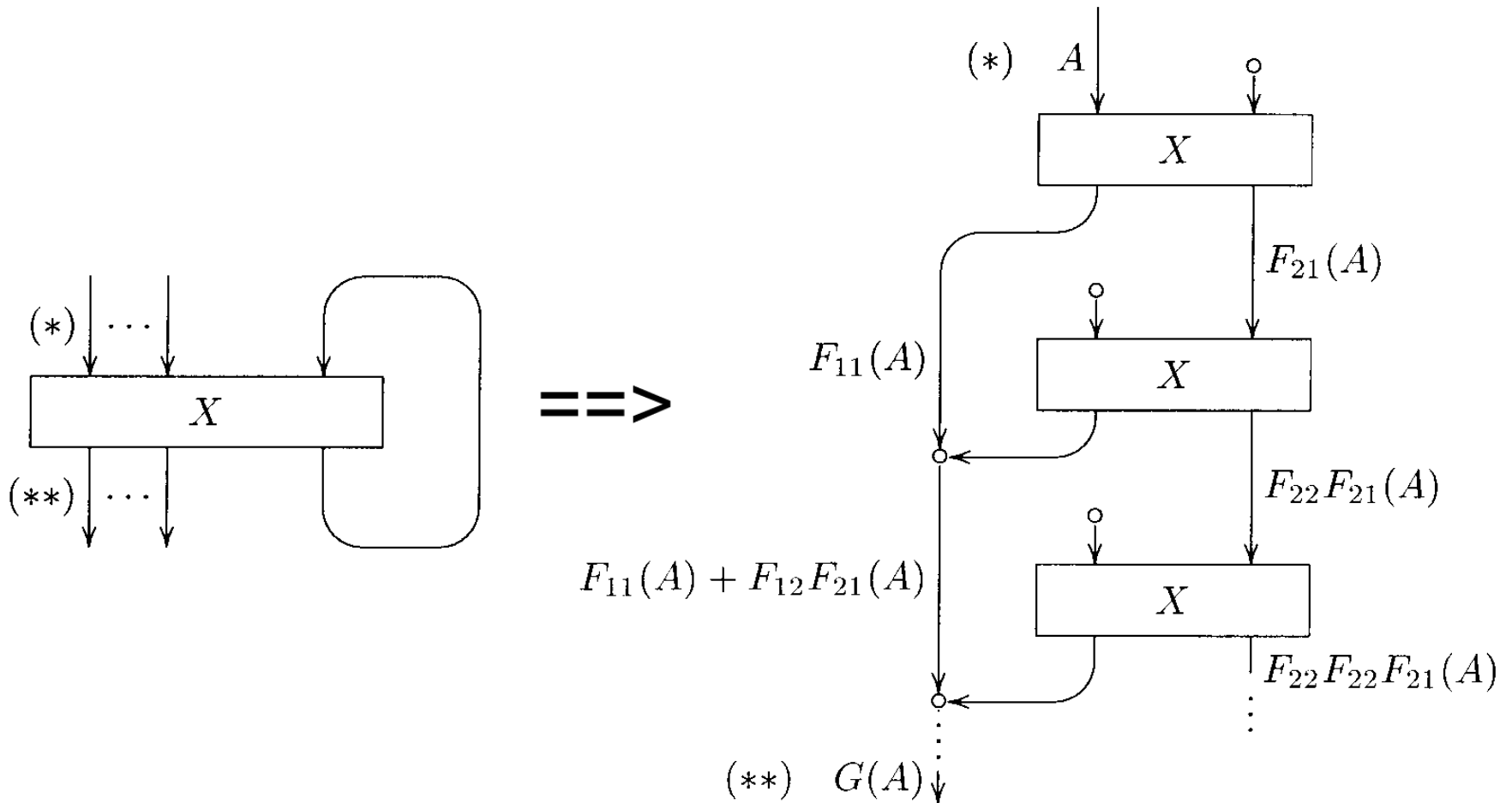
$\dots \rightarrow S'_1 \mid \text{pop}; \text{merge}; c \mid S'_0 : D$

$\rightarrow (S'_0, S'_1) \mid \text{merge}; c \mid D$

$\rightarrow S'_0 + S'_1 \mid c \mid D$

Looping

Semantics of a loop = Infinite unwind



Loop semantics

- Given $A = (A_1, \dots, A_n)$.
- Suppose semantics of X are $F(A_1, \dots, A_n, B) = (C_1, \dots, C_m, D)$.

Then

$$F(A, 0) = (F_{11}(A), F_{21}(A))$$

$$F(0, B) = (F_{12}(B), F_{22}(B))$$

and

$$G(A) = F_{11}(A) + \sum_{i=0}^{\infty} F_{12}(F_{22}^i(F_{21}(A)))$$

Looping in the quantum stack machine

Consider $\mathbb{I}\mathbb{L}(A) = A^{\mathbb{N}}$. Then, we add a *loop* instruction to the stack machine that has type:

$$qStack \times InsStream \times Dump \rightarrow \mathbb{I}\mathbb{L}(qStack \times InsStream \times Dump)$$

Recalling that $\mathbb{I}\mathbb{L}(_)$ is a monad, with

$$\eta(a) = \lambda n.a$$

$$\mu = diagonal$$

we can now consider our quantum stack machine in the Kleisli category, lifting the previously mentioned functions in the standard way.

- ⑥ Extensions to the types
 - △ Add tuples, sums and inductive types.
 - △ Add built-in types such as ints, chars.
- ⑥ Consider performance issues.
- ⑥ Investigate ways to handle IO of classical values.

Thanks

- ⑥ Robin Cockett, for many ideas and much encouragement
- ⑥ Peter Selinger, for the original paper (and the diagrams)
- ⑥ The Fields Summer School organizers.
- ⑥ My fellow grad students at Calgary.