Compiling a Quantum Programming Language

Brett Giles

gilesb@cpsc.ucalgary.ca

University of Calgary

Quantum vs. Classical

	Quantum	Classical
Data:	$\mathbf{qbit}$	$\mathbf{bit}$
Form:	$\alpha\mathbf{0} + \beta\mathbf{1}$ $\begin{pmatrix} \alpha\bar{\alpha} & \alpha\bar{\beta} \\ \beta\bar{\alpha} & \beta\bar{\beta} \end{pmatrix}$	0 or 1 $(a, b)(P(=0) = a)$
Operations:	Unitary Transforms	Logical Gates
Viewing:	Measure (collapses)	Branch
Multiples:	Entanglement	Control Flow

Measurement

- One quantum bit:

$$
\begin{tikzcd}
& \alpha\mathbf{0} + \beta\mathbf{1} \arrow[dl, "P_0{=}|\alpha|^2"', "\mathbf{0}"] \arrow[dr, "P_1{=}|\beta|^2", "\mathbf{1}"'] & \\
\mathbf{0} & & \mathbf{1}
\end{tikzcd}
$$

- Two q-bits, measure FIRST one:

$$
\begin{tikzcd}
& \alpha\mathbf{00} + \beta\mathbf{01} + \gamma\mathbf{10} + \delta\mathbf{11} \arrow[dl, "P_0{=}|\alpha|^2{+}|\beta|^2"', "\mathbf{0}"] \arrow[dr, "P_1{=}|\gamma|^2{+}|\delta|^2", "\mathbf{1}"'] & \\
\alpha\mathbf{00} + \beta\mathbf{01} & & \gamma\mathbf{10} + \delta\mathbf{11}
\end{tikzcd}
$$

Unitary Transformation, pure and mixed states

- A matrix $S \in \mathbb{C}^{n\times n}$is *Unitary* when $S^*S = I$. A unitary transformation will be represented by a Unitary matrix (S). It is applied to a set of $\mathbf{qbits}$ (represented by a matrix U) by applying this way: SUS^*.
- *Pure state*: Quantum system is described by the state vector $u \in \mathbb{C}^{2^n}$.
- *Mixed state*: an outside observer has the viewpoint that the system is in state u_i with probability λ_i. Denoted as the mixed state:

$$\lambda_1\{u_1\} + \cdots + \lambda_m\{u_m\}, \qquad \sum_i \lambda_i = 1$$

Block QPL

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

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

Block QPL Statements

Statements $P, Q ::=$

$\textbf{new bit } b := 0 \mid \textbf{new qbit } q := 0$

$\mid b := 0 \mid b := 1$

$\mid q_1, \ldots, q_n {*}= S$

$\mid \textbf{skip}$

$\mid B$

$\mid \textbf{if } b \textbf{ then } P \textbf{ else } Q \mid \textbf{measure } q \textbf{ then } P \textbf{ else } Q$

$\mid \textbf{while } b \textbf{ do } P$

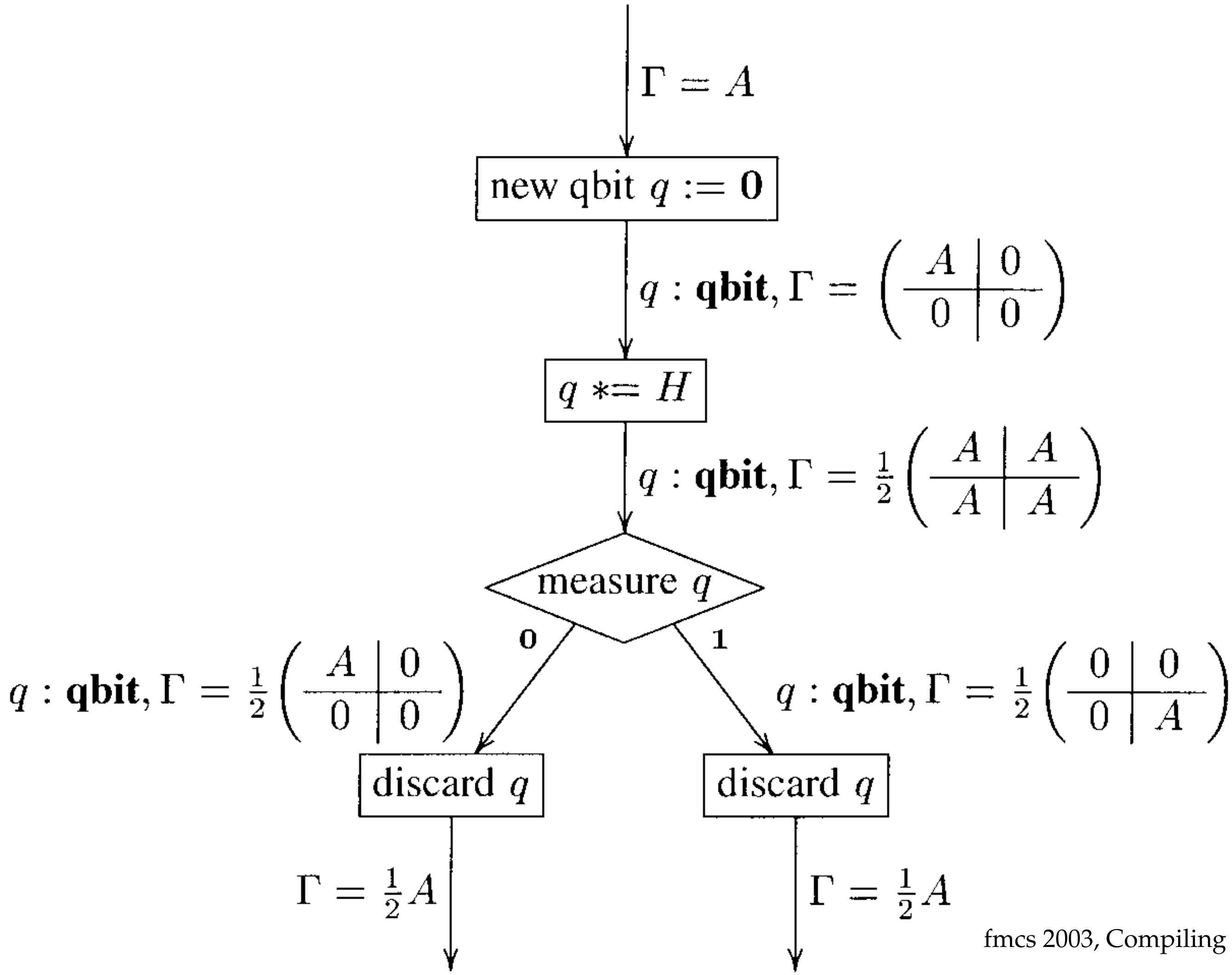
$\mid \textbf{import proc } X : \Gamma \to \Gamma \textbf{ in } Q$

$\mid \textbf{proc } X : \Gamma \to \Gamma \; \{P\} \textbf{ in } Q$

$\mid \textbf{call } X(x_1, \ldots, x_n)$

Examples of quantum flow charts

Fair Coin Toss



Example - a quantum coinflip.

```
{  proc cf : a:bit -> a:bit
    {
      new qbit q := 0;
      q *= H;
      measure q then
         a := 0
      else
         a := 1;
    } in
    {
      new bit x := 0;
      call cf(x);
      while x do
        call cf(x);
    }
}
```

Example - adding two bits.

```
export proc addwcarry:
        r:bit,carry:bit,a:bit,b:bit
        -> r:bit,carry:bit,a:bit,b:bit
{    carry:=0;
     if a then
        if b then {
           r := 0;
           carry := 1;
        }
        else
             r := 1;
     else
       if b then
          r:= 1;
       else
          r:= 0;
  }
```

Emulating the Quantum Machine

We considered two possible ways to do this:

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

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

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 $\mathbf{qbit}$s may not be copied. For example, when doing a unitary transform on 2 $\mathbf{qbit}$s, we may not use the same $\mathbf{qbit}$. As another example, when calling a procedure with more than one $\mathbf{qbit}$, they must all be distinct.

Combining classical and quantum data in Quantum Flow Charts

- Recall we only have two types, $\mathbf{bit}$ and $\mathbf{qbit}$, with typing contexts.
- Semantically, an edge labeled with n bits and m qbits can be replaced by 2^n edges each labeled with m qbits.
- The state for the above is a 2^n-tuple $(A_0, \ldots, A_{2^n-1})$ of density matrices each in $\mathbb{C}^{m\times m}$
- 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 $\rho = n_1, \ldots, n_s$.) We may associate a complex vector space

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

Then consider the category $\mathbb{V}$:

Objects: Signatures

Maps: $f : \rho \to \rho' \iff f$ is a complex linear function $f : V_\rho \to V_\rho'$

Identity: Identity function

Composition: Inherited

Semantics of QPL, cont'd

A *superoperator* is a linear function F that:

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

$[\![\text{new bit } b := 0]\!]$ $\qquad newbit :: \mathbf{I} \to \mathbf{bit} : a \mapsto (a, 0)$

$[\![\text{new qbit } q := 0]\!]$ $\qquad newqbit :: \mathbf{I} \to \mathbf{qbit} : a \mapsto \begin{pmatrix} a & 0 \\ 0 & 0 \end{pmatrix}$

$[\![\text{discard } b]\!]$ $\qquad discardbit :: \mathbf{bit} \to \mathbf{I} : (a, b) \mapsto a + b$

$[\![\text{discard } q]\!]$ $\qquad discardqbit :: \mathbf{qbit} \to \mathbf{I} : \begin{pmatrix} a & b \\ c & d \end{pmatrix} \mapsto a + d$

$[\![\text{measure } q]\!]$ $\qquad measure :: \mathbf{qbit} \to \mathbf{qbit} \oplus \mathbf{qbit} :$

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

A Quantum stack machine

- A standard machine to implement simple classical language is stack based.
- We use a tree as a “stack”, where each item in the stack is either a $\mathbf{bit}$ (has two branches) or $\mathbf{qbit}$ (has four branches).
- Each branch has a value associated with it (= probability in $\mathbf{bit}$, = elements of density matrix in $\mathbf{qbit}$.) A zero implies we do not need to save anything under that branch.

Quantum Stack machine instructions

$newbit, discardbit, if,$ $setbit, unsetbit$	$\mathbf{bit}$ operations
$newqbit, discardqbit,$ $measure, utrans(8)$	$\mathbf{qbit}$ operations
$merge, initial, pullup, ret$	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

$$S \mid pullup(b); \langle c_0 | c_1 \rangle; c \mid D$$

$$\to ((p_0, S_0), (p_1, S_1)) \mid \langle c_0 | c_1 \rangle; c \mid D$$

$$\to S_0 \mid c_0 \mid cond_0(p_0, p_1, S_1, c_1, c) : D$$

$$\cdots \to S_0' \mid ret \mid cond_0(p_0, p_1, S_1, c_1, c) : D$$

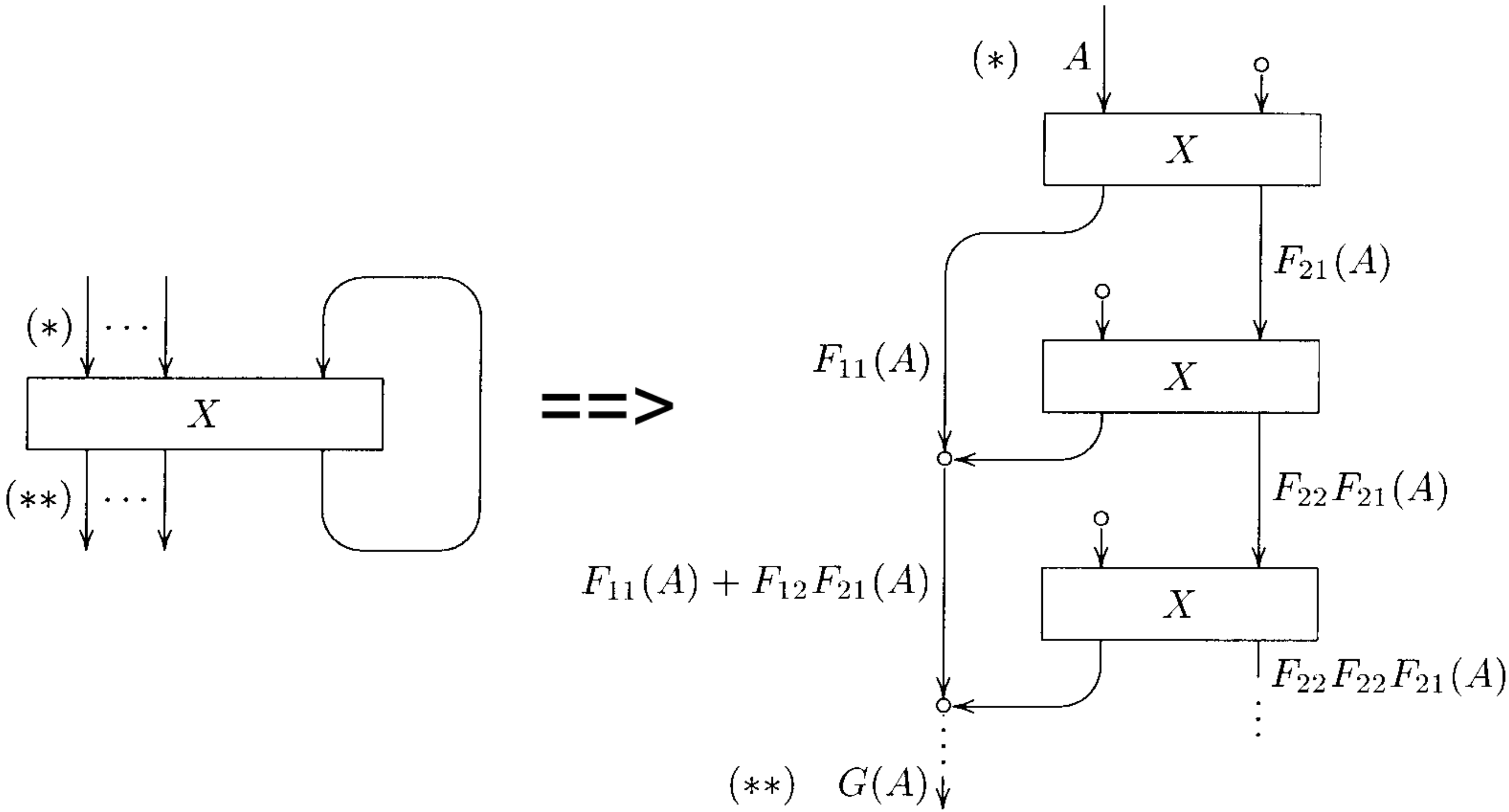
$$\to S_1 \mid c_1 \mid cond_1(p_0, p_1, S_0', c) : D$$

$$\cdots \to S_1' \mid ret \mid cond_1(p_0, p_1, S_0', c) : D$$

$$\to ((p_0, S_0'), (p_1, S_1')) \mid c \mid D$$

Looping

Semantics of a loop = Infinite unwind



Loop semantics

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

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

$$qStack \times InsStream \times Dump \to \mathbb{IL}(qStack \times InsStream \times Dump)$$

Recalling that $\mathbb{IL}(\_)$ 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.

Futures

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

Thanks

- Robin Cockett, for many ideas and much encouragement
- Peter Selinger, for the original paper (and the diagrams)
- The FMCS 2003 organizers.
- My fellow grad students at Calgary.