Quantum Languages

Eric Bond

Spring 2018

Outline

- Motivation
  - Analogies from current language design
- Dichotomy of research directions
  - Compiler level
    - Optimizing Assembly
  - Programming language level
    - Optimizing usability
    - Exploring alternative mathematical perspectives
- Survey of existing languages
  - Quipper
  - Q#
  - Prior work

Why?

```
        .file   "demo.c"
        .text
        .globl  tree
        .type   tree, @function
tree:
.LFB2:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $48, %rsp
        movl    %edi, -20(%rbp)
        movq    %rsi, -32(%rbp)
        movq    %rdx, -40(%rbp)
        movl    $24, %edi
        call    malloc
        movq    %rax, -8(%rbp)
        movq    -8(%rbp), %rax
        movl    -20(%rbp), %edx
        movl    %edx, (%rax)
        movq    -8(%rbp), %rax
        movq    -32(%rbp), %rdx
        movq    %rdx, 8(%rax)
        movq    -8(%rbp), %rax
        movq    -40(%rbp), %rdx
        movq    %rdx, 16(%rax)
        movq    -8(%rbp), %rax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.L6:
        nop
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE3:
        .size   inorder, .-inorder
        .globl  main
        .type   main, @function
.LFE2:
        .size   tree, .-tree
        .section .rodata
.LC0:
        .string "%d\n"
        .text
        .globl  inorder
        .type   inorder, @function
inorder:
.LFB3:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movq    %rdi, -8(%rbp)
        movq    -8(%rbp), %rax
        movq    8(%rax), %rax
        testq   %rax, %rax
        je      .L4
        movq    -8(%rbp), %rax
        movq    8(%rax), %rax
        movq    %rax, %rdi
        call    inorder
.L4:
        movq    -8(%rbp), %rax
        movl    (%rax), %eax
        movl    %eax, %esi
        movl    $.LC0, %edi
        movl    $0, %eax
        call    printf
        movq    -8(%rbp), %rax
        movq    16(%rax), %rax
        testq   %rax, %rax
        je      .L6
        movq    -8(%rbp), %rax
        movq    16(%rax), %rax
        movq    %rax, %rdi
        call    inorder

main:
.LFB4:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE4:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1
        .section .note.GNU-stack,"",@progbits
```

x64 Assembly

```c
#include <stdlib.h>
#include <stdio.h>

typedef struct node_s
{
  int value;
  struct node_s* left;
  struct node_s* right;
} *node;

node tree(int v, node l, node r)
{
  node n = malloc(sizeof(struct node_s));
  n->value = v;
  n->left  = l;
  n->right = r;
  return n;
}

void inorder(node n)
{
  if (n->left)
    inorder(n->left);
  printf("%d\n",n->value);
  if (n->right)
    inorder(n->right);
}

int main(){}
```

C

```haskell
data Tree a = Leaf | Node a (Tree a) (Tree a)

inorder (Node v l r) = inorder l ++ (v : inorder r)
inorder Leaf = []
```

Haskell

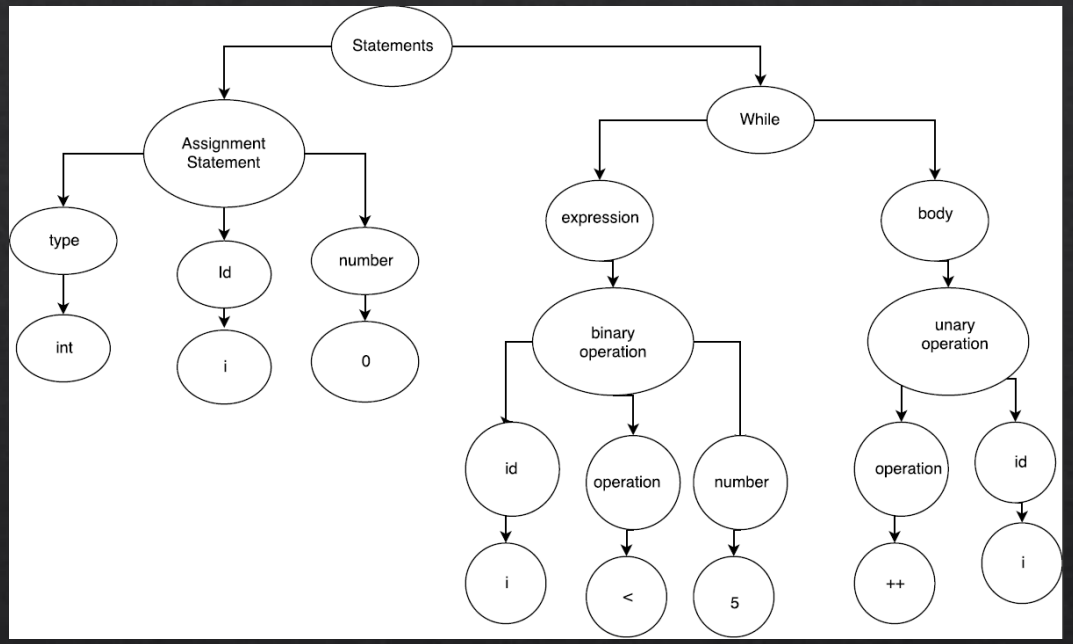
Research Direction - Compilers

- Classical Compilers

```
int i = 0;
while ( i < 5 )
{
    i ++;
}
```

```
<type> <id> <equal> <number> <semicolon>

<while> <L-paren> <id> <less-than> <number> <R-paren>
<L-brace>
    <id> <plus> <plus> <semicolon>
<R-brace>
```

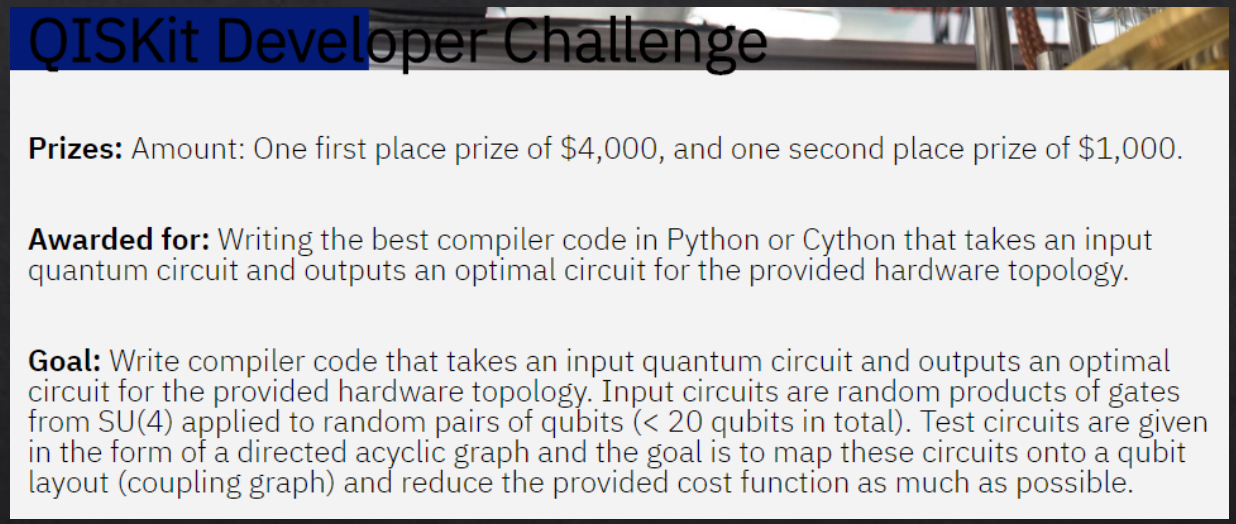


Optimization

Code generation

Research Direction - Compilers

- Quantum Compilers – current and potential features
  - Decomposition of arbitrary unitary matrices
  - Optimal assembly code generation
    - Optimize for qubit chip topology
    - Algebraic rewrite rules
    - Single gate transforms
      - (u1 u2 u3 gates in QASM)
    - Qubit allocation and reuse
      - (modified graph coloring)
  - Error correction and decoherence prevention
    - Problem: physical vs logical qubit?
      - Steane codes
      - Qubit redundancy
- Next steps?
  - Representation theory?



Research Directions – PL & Theory

- Higher abstraction in QPL
  - How to adopt superposition and entanglement as language primitives?
  - What kinds of data types or data structures can we construct with qubits?
  - What are frequently occurring patterns in quantum algorithm design?
    - How to incorporate these patterns in language design?
  - How will quantum machines interface with classical machines programmatically?

- How to reason about quantum circuits?

Programming Languages as Formal Systems

```
Variable P : PreCategory -> Type.
Context `{forall C, IsHProp (P C)}.
Context `{HF : forall C D, P C -> P D -> IsHSet (Functor C D)}.

Lemma is_terminal_object__is_terminal_category
      `(IsTerminalCategory one)
      (HT : P one)
: IsTerminalObject (sub_pre_cat P HF) (one; HT).
Proof.
  typeclasses eauto.
Defined.

Lemma is_initial_object__is_initial_category
      `(IsInitialCategory zero)
      (HI : P zero)
: IsInitialObject (sub_pre_cat P HF) (zero; HI).
Proof.
  typeclasses eauto.
Defined.
```

Coq

Source: https://github.com/HoTT/HoTT/blob/master/theories/Categories/Cat/Core.v

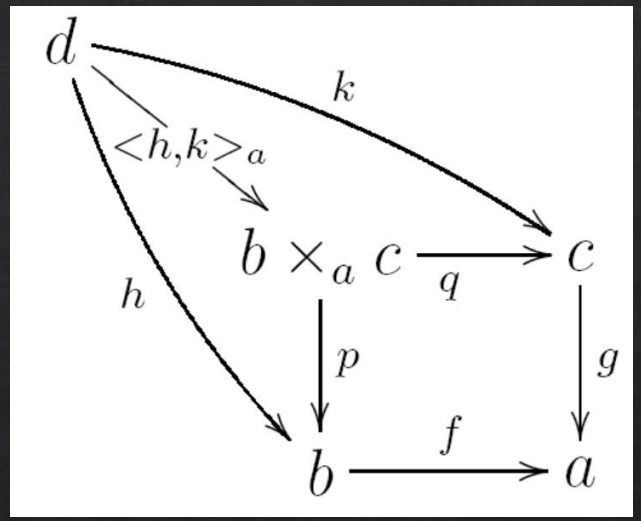
```
isomorphism-preserves-infinite-cyclic : ∀ {i j}
  → {⊙G : ⊙Group i} {⊙H : ⊙Group j} → ⊙G ⊙≃ᴳ ⊙H
  → is-infinite-cyclic ⊙G
  → is-infinite-cyclic ⊙H
isomorphism-preserves-infinite-cyclic
  {⊙G = ⊙[ G , g ]ᴳ} {⊙H = ⊙[ H , h ]ᴳ} (iso , pres-pt) ⊙G-iic =
  is-eq _ (is-equiv.g ⊙G-iic ∘ GroupIso.g iso) to-from from-to
  where
    abstract
      lemma : Group.exp H h ∼ GroupIso.f iso ∘ Group.exp G g
      lemma i = ! $
        GroupIso.f iso (Group.exp G g i)
          =⟨ GroupIso.pres-exp iso g i ⟩
        Group.exp H (GroupIso.f iso g) i
          =⟨ ap (λ x → Group.exp H x i) pres-pt ⟩
        Group.exp H h i
          =■
```

Agda

Source : https://github.com/HoTT/HoTT-Agda/blob/master/theorems/groups/Pointed.agda

Higher Abstraction for Quantum Computing

- First approximation
  - Applied Category Theory?
    - "As a first approximation, one could say that category theory is the mathematical study of the abstract algebras of functions“ - Awodey
  - Linear Logic?
    - Captures the notion of restriction of resource usage
    - Used to model measurement and no-cloning rules



Category Theory	Physics	Topology	Logic	Computation
object	system	manifold	proposition	data type
morphism	process	cobordism	proof	program

Current Philosophy

- Quantum as a co-processor
- High level classical languages generate quantum assembly

```
include "qelib1.inc";
qreg q[5];
creg c[5];

h q[0];
h q[1];
h q[2];
h q[1];
z q[0];
cx q[2],q[1];
h q[1];
h q[0];
h q[1];
h q[2];
measure q[0] -> c[0];
measure q[1] -> c[1];
measure q[2] -> c[2];
```

??

Quipper

- Circuit description language
  - Model: Symmetric monoidal category
  - Has linear types
- Aims for a unified model for parameters and state
  - Parameter: a value known at circuit generation time
  - State: a value known at circuit execution time

```
function deutsch_jozsa_circuit :: Oracle → Circ [Bit]

deutsch_jozsa_circuit oracle = do

        top_qubits ← qinit (replicate (qubit_num oracle) False)
        bottom_qubit ← qinit True

        mapUnary hadamard top_qubits
        hadamard_at bottom_qubit

        function oracle (top_qubits, bottom_qubit)

        mapUnary hadamard top_qubits

        (top_qubits, bottom_qubit) ← measure (top_qubits, bottom_qubit)

        cdiscard bottom_qubit
        return top_qubits
```

Quipper – quantum programming
Source: arXiv:1406.4481 **[quant-ph]**

Q#

- Microsoft's work
- Embraces co-processor model
- 3 levels of computation
  - Classical computation that reads input data, sets up the quantum computation, triggers the quantum computation, processes the results of the computation, and presents the results to the user.
  - Quantum computation that happens directly in the quantum device and implements a quantum algorithm.
  - Classical computation that is required by the quantum algorithm during its execution.

References and Further reading

- Categorical model for a quantum circuit description language – Rios & Selinger
- Category theory – Awodey
- Quantum programming languages: survey and bibliography – Gay
- A taste of linear logic – Wadler
- Physics topology logic computation : a rosetta stone – Baez
- Categorical models for quantum computing - Wester
- Open quantum assembly language – Cross, Bishop, Smolin, Gambetta
- 5 quantum algorithms using quipper - Selinger
- Categorical quantum mechanics – Abramsky - Coecke
- Q# website