Sized Types for low-level Quantum Metaprogramming

Matthew Amy

University of Waterloo, Waterloo, Canada

Reversible Computation
June 25th, 2019

Quantum libraries & Standardization

...additional work will be needed on the sorts of modularity and layering commonly needed for scalable systems. For example, libraries for commonly-used functions will aid development and optimization...

— Martonosi & Roetteler, CCC report on Next Steps in Quantum Computing

We identified a joint desire for a rich, but machine readable, intermediate language...

— Thomas Parks on the Oxford NQIT First Meeting for the Strategic Initiative in Quantum Software

Circuit families

Basic unit of a quantum circuit library is a **circuit family**

- E.g. arithmetic or algorithm in arbitrary bit sizes
- More efficient to send circuits in batches to QPU
- Easier & better optimization available

Circuit families

Basic unit of a quantum circuit library is a **circuit family**

- E.g. arithmetic or algorithm in arbitrary bit sizes
- More efficient to send circuits in batches to QPU
- Easier & better optimization available

Balancing expressivity of circuit generation with ease of use & efficiency is hard!

Host-language metaprogramming

a.k.a. embedded circuit description languages

Standard solution uses convenient features of a **host** language to program **generators** for circuit families

E.g. in Quipper

```
qft_adder :: [Qubit] -> [Qubit] -> Circ ()
qft_adder _ [] = return ()
qft_adder as (b:bs) = do
    qft_adder' as b 1
    qft_adder (tail as) bs
  where
    qft_adder' :: [Qubit] -> Qubit -> Int -> Circ [Qubit]
    qft_adder' [] _ _ = return []
    qft_adder' (a:as) b n = do
      b <- controlled (rGate n b) a
      qft_adder' as b (n+1)
```

Pros: immediately have rich syntax & set of programming tools
Cons: not easily portable & restricts features

Metaprogramming in standalone languages

- Typically only in higher-level languages
- Usually via dynamic-length arrays and lists

E.g. in Q#:

```
operation AddI (
  xs : Microsoft.Quantum.Arithmetic.LittleEndian,
  ys : Microsoft.Quantum.Arithmetic.LittleEndian) : Unit
{ ... }
```

Metaprogramming in standalone languages

- Typically only in higher-level languages
- Usually via dynamic-length arrays and lists

E.g. in Q#:

```
operation AddI (
  xs : Microsoft.Quantum.Arithmetic.LittleEndian,
  ys : Microsoft.Quantum.Arithmetic.LittleEndian) : Unit
{ ... }
```

Problems:

Metaprogramming in standalone languages

- Typically only in higher-level languages
- Usually via dynamic-length arrays and lists

E.g. in Q#:

```
operation AddI (
  xs : Microsoft.Quantum.Arithmetic.LittleEndian,
  ys : Microsoft.Quantum.Arithmetic.LittleEndian) : Unit
{ ... }
```

Problems:

- run-time error if `ys` is smaller than `xs`

Metaprogramming in standalone languages

- Typically only in higher-level languages
- Usually via dynamic-length arrays and lists

E.g. in Q#:

```
operation AddI (
  xs : Microsoft.Quantum.Arithmetic.LittleEndian,
  ys : Microsoft.Quantum.Arithmetic.LittleEndian) : Unit
{ ... }
```

Problems:

- run-time error if `ys` is smaller than `xs`
- no **semantically consistent & concise** method of generating a circuit specialized to particular sizes

metaQASM

A low-level circuit description language extending the Quantum Assembly Language (QASM) with **explicit** specification and specialization of size-parametrized circuit families

Features:

- Lightweight dependent types (sized types) in the vein of Dependent ML/ATS[1]
- Statically rules out register out-of-bounds accesses
- Metaprogramming-free fragment allows more concise & expressive QASM programs

[1] Xi, *Dependent ML An approach to practical programming with dependent types*

QASM as an intermediate language

We identified a joint desire for a rich, but machine readable, intermediate language... the format of QASM was viewed positively but a need for extensible classical coroutines and the ability to express large indivisible unitary gates is needed to embrace OpenQASM as a general standard.

— Thomas Parks on the Oxford NQIT First Meeting for the Strategic Initiative in Quantum Software

openQASM specification

```
⟨mainprogram⟩ ⊨ OPENQASM ⟨real⟩ ; ⟨program⟩
    ⟨program⟩ ⊨ ⟨statement⟩ | ⟨program⟩ ⟨statement⟩
  ⟨statement⟩ ⊨ ⟨decl⟩
                | ⟨gatedecl⟩ ⟨goplist⟩ }
                | ⟨gatedecl⟩ }
                | opaque ⟨id⟩ ⟨idlist⟩ ;
                | opaque ⟨id⟩ ( ) ⟨idlist⟩ ; | opaque ⟨id⟩ ( ⟨idlist⟩ ) ⟨idlist⟩ ;
                | ⟨qop⟩
                | if ( ⟨id⟩ == ⟨nninteger⟩ ) ⟨qop⟩
                | barrier ⟨anylist⟩ ;
       ⟨decl⟩ ⊨ qreg ⟨id⟩ [ ⟨nninteger⟩ ] ; | creg ⟨id⟩ [ ⟨nninteger⟩ ] ;
   ⟨gatedecl⟩ ⊨ gate ⟨id⟩ ⟨idlist⟩ {
                | gate ⟨id⟩ ( ) ⟨idlist⟩ {
                | gate ⟨id⟩ ( ⟨idlist⟩ )⟨idlist⟩ {
    ⟨goplist⟩ ⊨ ⟨uop⟩
                | barrier ⟨idlist⟩ ;
                | ⟨goplist⟩ ⟨uop⟩
                | ⟨goplist⟩ barrier ⟨idlist⟩ ;
        ⟨qop⟩ ⊨ ⟨uop⟩
                | measure ⟨argument⟩ - > ⟨argument⟩ ;
                | reset ⟨argument⟩ ;

        ⟨uop⟩ ⊨ U ( ⟨explist⟩ ) ⟨argument⟩ ;
                | CX ⟨argument⟩ , ⟨argument⟩ ;
                | ⟨id⟩ ⟨anylist⟩ ; | ⟨id⟩ ( ) ⟨anylist⟩ ;
                | ⟨id⟩ ( ⟨explist⟩ ) ⟨anylist⟩ ;
    ⟨anylist⟩ ⊨ ⟨idlist⟩ | ⟨mixedlist⟩
     ⟨idlist⟩ ⊨ ⟨id⟩ | ⟨idlist⟩ , ⟨id⟩
  ⟨mixedlist⟩ ⊨ ⟨id⟩ [ ⟨nninteger⟩ ] | ⟨mixedlist⟩ , ⟨id⟩
                | ⟨mixedlist⟩ , ⟨id⟩ [ ⟨nninteger⟩ ]
                | ⟨idlist⟩ , ⟨id⟩[ ⟨nninteger⟩ ]

   ⟨argument⟩ ⊨ ⟨id⟩ | ⟨id⟩ [ ⟨nninteger⟩ ]
    ⟨explist⟩ ⊨ ⟨exp⟩ | ⟨explist⟩ , ⟨exp⟩
        ⟨exp⟩ ⊨ ⟨real⟩ | ⟨nninteger⟩ | pi | ⟨id⟩
                | ⟨exp⟩ + ⟨exp⟩ | ⟨exp⟩ - ⟨exp⟩ | ⟨exp⟩ * ⟨exp⟩
                | ⟨exp⟩ / ⟨exp⟩ | - ⟨exp⟩ | ⟨exp⟩ ^ ⟨exp⟩
                | ( ⟨exp⟩ ) | ⟨unaryop⟩ ( ⟨exp⟩ )

    ⟨unaryop⟩ ⊨ sin | cos | tan | exp | ln | sqrt
```

Informal specifications:

- Non-parenthesized arguments to gate must be quantum types
- Quantum arguments can not be dereferenced in gate body
- A gate applied to a register is mapped to each qubit of the register

Teleportation example

```
OPENQASM 2.0;
include "qelib1.inc";

qreg q[3];
creg c0[1];
creg c1[1];

h q[1];
cx q[1],q[2];
cx q[0],q[1];
h q[0];
measure q[0] -> c0[0];
measure q[1] -> c1[0];
if(c0==1) z q[2];
if(c1==1) x q[2];
```

typedQASM

$$
\begin{array}{rcl}
\text{Base types } \beta & ::= & \texttt{Bit} \mid \texttt{Qbit} \\
\text{Types } \tau & ::= & \beta \mid \beta[I] \mid \texttt{Circuit}(\tau_1, \ldots, \tau_n) \\
\\
\text{Index } I & ::= & i \in \mathbb{N} \\
\text{Expression } E & ::= & x \mid x[I] \\
\text{Unitary Stmt } U & ::= & \texttt{cx}(E_1, E_2) \mid \texttt{h}(E) \mid \texttt{t}(E) \mid \texttt{tdg}(E) \mid E(E_1, \ldots, E_n) \mid U_1\texttt{;}\ U_2 \\
\text{Command } C & ::= & \texttt{creg}\ x[I] \mid \texttt{qreg}\ x[I] \\
 & \mid & \texttt{gate}\ x(x_1 : \tau_1, \ldots, x_n : \tau_n)\ \texttt{\{}\ U\ \texttt{\}} \\
 & \mid & \texttt{measure}\ E_1\ \texttt{->}\ E_2 \mid \texttt{reset}\ E \mid U \\
 & \mid & \texttt{if(}E\texttt{==}I\texttt{)}\ \texttt{\{}\ U\ \texttt{\}} \mid C_1\texttt{;}\ C_2
\end{array}
$$

- Supports registers as parameters
- Supports gates as parameters
- Other features of openQASM easy to add in (barriers, uniform circuits, classical parameters, etc.)

typedQASM semantics

A configuration $\langle S, \sigma, \eta, |\psi\rangle\rangle$ consists of

- S – syntactic element
- σ – environment
- η – classical state
- $|\psi\rangle$ – quantum state

$$\frac{x \in \mathrm{dom}(\sigma)}{\langle x, \sigma, \eta, |\psi\rangle\rangle \Downarrow \sigma(x)} \qquad \frac{\langle x, \sigma, \eta, |\psi\rangle\rangle \Downarrow (l_0, \ldots, l_{I'}) \qquad I \leq I'}{\langle x[I], \sigma, \eta, |\psi\rangle\rangle \Downarrow l_I}$$

$$\frac{\langle E_1, \sigma, \eta, |\psi\rangle\rangle \Downarrow l_1 \qquad \langle E_2, \sigma, \eta, |\psi\rangle\rangle \Downarrow l_2}{\langle \texttt{measure}\ E_1\ \texttt{->}\ E_2, \sigma, \eta, |\psi\rangle\rangle \Downarrow \langle \sigma, \eta[l_2 \leftarrow 0], P^0_{l_1}|\psi\rangle\rangle}$$

$$\frac{\langle E_1, \sigma, \eta, |\psi\rangle\rangle \Downarrow l_1 \qquad \langle E_2, \sigma, \eta, |\psi\rangle\rangle \Downarrow l_2}{\langle \texttt{measure}\ E_1\ \texttt{->}\ E_2, \sigma, \eta, |\psi\rangle\rangle \Downarrow \langle \sigma, \eta[l_2 \leftarrow 1], P^1_{l_1}|\psi\rangle\rangle}$$

Possible errors:

- Undefined variable (hard)
- "Regular" type errors (hard or soft)
- Out-of-bounds dereference (hard)

Type system

The Obvious Type System$^{\text{TM}}$ (+ subtyping on register sizes)

$$\frac{\Gamma \vdash x : \beta[I'] \qquad I \leq I' - 1}{\Gamma \vdash x[I] : \beta} \qquad \frac{\Gamma \vdash E : \beta[I'] \qquad I \leq I'}{\Gamma \vdash E : \beta[I]}$$

Theorem (Normalization)

If $\vdash C : \texttt{Unit}$, *then*

$$\langle C, \emptyset, \lambda I.0, |00\cdots\rangle\rangle \Downarrow \langle \sigma, \eta, |\psi\rangle\rangle.$$

metaQASM

$$
\begin{array}{rcl}
\text{Range } \iota & ::= & [I_1, I_2] \\
\text{Types } \tau & ::= & \ldots \mid \texttt{Family}(y_1, \ldots, y_m)(\tau_1, \ldots, \tau_n) \\
\\
\text{Index } I & ::= & \ldots \mid y \mid \infty \mid I_1 + I_2 \mid I_1 - I_2 \mid I_1 \cdot I_2 \\
\text{Expression } E & ::= & \ldots \mid \texttt{instance}(I_1, \ldots, I_m)\ E \\
\text{Unitary Stmt } U & ::= & \ldots \mid \texttt{reverse}\ U \\
 & & \mid \texttt{for}\ y = I_1..I_2\ \texttt{do}\ \{\ U\ \} \\
\text{Command } C & ::= & \ldots \mid \texttt{family}(y_1, \ldots, y_m)\ x(x_1 : \tau_1, \ldots, x_n : \tau_n)\ \{\ U\ \}\ \texttt{in}\ \{\ C\ \}
\end{array}
$$

Extends typedQASM with

- gate inversion
- for loops
- **index abstraction, expressions & application**

Example

```
include "toffoli.qasm";

gate maj(a:Qbit, b:Qbit, c:Qbit, res:Qbit) {
    toffoli(b, c, res);
    cx(b, c);
    toffoli(a, c, res);
    cx(b, c)
}

family(n) add(a:Qbit[n], b:Qbit[n], c:Qbit[n], anc:Qbit[n]) {
    cx(a[0], c[0]);
    cx(b[0], c[0]);
    toffoli(a[0], b[0], anc[0]);
    for i=1..n-1 do {
        cx(a[i], c[i]);
        cx(b[i], c[i]);
        cx(anc[i-1], c[i]);
        maj(a[i], b[i], anc[i-1], anc[i])
    }
}
```

More examples

Multiplication:

```
family(n) mult(x:Qbit[n], y:Qbit[n], z:Qbit[2*n], anc:Qbit,
    ctrlAdd:Family(m)(x:Qbit, y:Qbit[m], z:Qbit[m], c:Qbit))
{
  for i=0..n-1 do {
    instance(n) ctrlAdd(x[i], y, z[i..i+n-1], anc)
  }
}
```

Quantum Fourier Transform:

```
include "cphase.qasm";
family(n) qft(x:Qbit[n]) {
    for i=0..n-1 do {
        h(x[i]);
        for j=i+1..n-1 do {
            cphase(j-1+1)(x[i], x[j])
        }
    }
}
```

Semantics

$$\frac{\langle E, \sigma, \eta, |\psi\rangle \Downarrow \Pi y_1, \ldots, y_m.\lambda x_1 : \tau_1, \ldots, x_n : \tau_n.U}{\langle \texttt{instance}(I_1, \ldots, I_m)\ \ E, \sigma, \eta, |\psi\rangle \Downarrow (\lambda x_1 : \tau_1, \ldots, x_n : \tau_n.U)\{I_1/y_1, \ldots, I_m/y_m\}}$$

$$\frac{\langle I_1, \sigma, \eta, |\psi\rangle\rangle \Downarrow i_1 \quad \langle I_2, \sigma, \eta, |\psi\rangle\rangle \Downarrow i_2 \quad i_1 > i_2}{\langle \texttt{for}\ \ y = I_1..I_2\ \ \texttt{do}\ \{\ \ U\ \ \}, \sigma, \eta, |\psi\rangle\rangle \Downarrow |\psi\rangle}$$

$$\frac{\begin{array}{c}\langle I_1, \sigma, \eta, |\psi\rangle\rangle \Downarrow i_1 \quad \langle I_2, \sigma, \eta, |\psi\rangle\rangle \Downarrow i_2 \quad i_1 \leq i_2 \\ \langle U\{i_1/y\}, \sigma, \eta, |\psi\rangle\rangle \Downarrow |\psi'\rangle \\ \langle \texttt{for}\ \ y = i_1 + 1..i_2\ \ \texttt{do}\ \{\ \ U\ \ \}, \sigma, \eta, |\psi'\rangle\rangle \Downarrow |\psi''\rangle\end{array}}{\langle \texttt{for}\ \ y = I_1..I_2\ \ \texttt{do}\ \{\ \ U\ \ \}, \sigma, \eta, |\psi\rangle\rangle \Downarrow |\psi''\rangle}$$

$$\frac{\langle U, \sigma, \eta, |\psi\rangle\rangle \Uparrow |\psi'\rangle}{\langle \texttt{reverse}\ \ U, \sigma, \eta, |\psi\rangle\rangle \Downarrow |\psi'\rangle}$$

Semantics (reversal)

$$\frac{\langle E, \sigma, \eta, |\psi\rangle\rangle \Downarrow l}{\langle \mathtt{h}(E), \sigma, \eta, |\psi\rangle\rangle \Uparrow H_l|\psi\rangle} \quad \frac{\langle E, \sigma, \eta, |\psi\rangle\rangle \Downarrow l}{\langle \mathtt{t}(E), \sigma, \eta, |\psi\rangle\rangle \Uparrow T_l^\dagger|\psi\rangle} \quad \frac{\langle E, \sigma, \eta, |\psi\rangle\rangle \Downarrow l}{\langle \mathtt{tdg}(E), \sigma, \eta, |\psi\rangle\rangle \Uparrow T_l|\psi\rangle}$$

$$\frac{\langle E_1, \sigma, \eta, |\psi\rangle\rangle \Downarrow l_1 \quad \langle E_2, \sigma, \eta, |\psi\rangle\rangle \Downarrow l_2}{\langle \mathtt{cx}(E_1, E_2), \sigma, \eta, |\psi\rangle\rangle \Uparrow \mathrm{CNOT}_{l_1,l_2}|\psi\rangle} \quad \frac{\begin{array}{c}\langle E, \sigma, \eta, |\psi\rangle\rangle \Downarrow \lambda x_1, \ldots, x_n.U, \\ \langle U\{E_1/x_1, \ldots, E_n/x_n\}, \sigma, \eta, |\psi\rangle\rangle \Uparrow |\psi'\rangle\end{array}}{\langle E(E_1, \ldots, E_n), \sigma, \eta, |\psi\rangle\rangle \Uparrow |\psi'\rangle}$$

$$\frac{\langle U_2, \sigma, \eta, |\psi\rangle\rangle \Uparrow |\psi'\rangle \qquad \langle U_1, \sigma, \eta, |\psi'\rangle\rangle \Uparrow |\psi''\rangle}{\langle U_1;\ U_2, \sigma, \eta, |\psi\rangle\rangle \Uparrow |\psi''\rangle}$$

$$\frac{\langle U, \sigma, \eta, |\psi\rangle\rangle \Downarrow |\psi'\rangle}{\langle \mathtt{reverse}\ U, \sigma, \eta, |\psi\rangle\rangle \Uparrow |\psi'\rangle} \quad \frac{\langle l_1, \sigma, \eta, |\psi\rangle\rangle \Downarrow i_1 \quad \langle l_2, \sigma, \eta, |\psi\rangle\rangle \Downarrow i_2 \quad i_2 > i_1}{\langle \mathtt{for}\ \ y = l_1..l_2\ \ \mathtt{do}\ \{\ U\ \}, \sigma, \eta, |\psi\rangle\rangle \Uparrow |\psi\rangle}$$

$$\frac{\begin{array}{c}\langle l_1, \sigma, \eta, |\psi\rangle\rangle \Downarrow i_1 \quad \langle l_2, \sigma, \eta, |\psi\rangle\rangle \Downarrow i_2 \quad i_2 \geq i_1 \\ \langle U\{i_2/y\}, \sigma, \eta, |\psi\rangle\rangle \Uparrow |\psi'\rangle \\ \langle \mathtt{for}\ \ y = i_1..i_2 - 1\ \ \mathtt{do}\ \{\ U\ \}, \sigma, \eta, |\psi'\rangle\rangle \Uparrow |\psi''\rangle\end{array}}{\langle \mathtt{for}\ \ y = l_1..l_2\ \ \mathtt{do}\ \{\ U\ \}, \sigma, \eta, |\psi\rangle\rangle \Uparrow |\psi''\rangle}$$

Type system

TypedQASM + Dependent ML

$$
\begin{array}{lcl}
\text{Range } \iota & ::= & [I_1, I_2] \\
\text{Types } \tau & ::= & \beta \mid \beta[I] \mid \texttt{Circuit}(\tau_1, \ldots, \tau_n) \\
 & \mid & \texttt{Family}(y_1, \ldots, y_m)(\tau_1, \ldots, \tau_n)
\end{array}
$$

Main points:

- Two sorts: **types** and **(integer) ranges**
- Quantification of **types** over **ranges** (`Family` type)
- Quantification has fixed bounds
  - In the sense of subtyping between ranges...
  - ...and all quantified variables have $\mathbb{N} = [0, \infty]$ range

Type system

TypedQASM + Dependent ML

$$
\begin{array}{lcl}
\text{Range } \iota & ::= & [l_1, l_2] \\
\text{Types } \tau & ::= & \beta \mid \beta[l] \mid \texttt{Circuit}(\tau_1, \ldots, \tau_n) \\
 & \mid & \texttt{Family}(y_1, \ldots, y_m)(\tau_1, \ldots, \tau_n)
\end{array}
$$

Main points:

- Two sorts: **types** and **(integer) ranges**
- Quantification of **types** over **ranges** (`Family` type)
- Quantification has fixed bounds
  - In the sense of subtyping between ranges...
  - ...and all quantified variables have $\mathbb{N} = [0, \infty]$ range

Roughly speaking, $\texttt{Family}(y_1, \ldots, y_m)(\tau_1, \ldots, \tau_n)$ is equivalent to the dependent product type

$$\Pi_{y_1:\mathbb{N}, \ldots, y_m:\mathbb{N}}.(\tau_1 * \cdots * \tau_n) \to \text{Unit}$$

Index typing

$$\frac{}{\Delta \vdash i : [i, i]} \qquad \frac{y : [I_1, I_2] \in \Delta}{\Delta \vdash y : [I_1, I_2]} \qquad \frac{\Delta \vdash I : [I_1, I_2] \quad \Delta \models I'_1 \leq I_1 \quad \Delta \models I'_2 \geq I_2}{\Delta \vdash I : [I'_1, I'_2]}$$

$$\frac{\Delta \vdash I : [I_1, I_2] \quad \Delta \vdash I' : [I'_1, I'_2]}{\Delta \vdash I + I' : [I_1 + I'_1, I_2 + I'_2]} \qquad \frac{\Delta \vdash I : [I_1, I_2] \quad \Delta \vdash I' : [I'_1, I'_2]}{\Delta \vdash I - I' : [I_1 - I'_1, I_2 - I'_2]}$$

$$\frac{\begin{array}{c}\Delta \vdash I : [I_1, I_2] \quad \Delta \vdash I' : [I'_1, I'_2] \\ \Delta \models I''_1 = \min(I_1 \cdot I'_1, I_1 \cdot I'_2, I_2 \cdot I'_1, I_2 \cdot I'_2) \\ \Delta \models I''_2 = \max(I_1 \cdot I'_1, I_1 \cdot I'_2, I_2 \cdot I'_1, I_2 \cdot I'_2)\end{array}}{\Delta \vdash I \cdot I' : [I''_1, I''_2]}$$

- Δ (index context) maps variables to range types
- Judgements $\Delta \models P$ for a predicate P state that in the theory of integer arithmetic, P holds under the constraints in Δ

Regular typing

$$\frac{\Delta; \Gamma \vdash x : \beta[I'] \qquad \Delta \models 0 \leq I < I'}{\Delta; \Gamma \vdash x[I] : \beta}$$

$$\frac{\begin{array}{c}\Delta; \Gamma \vdash E : \mathtt{Family}(y_1, \ldots, y_m)(\tau_1, \ldots, \tau_n) \\ \Delta \vdash I_1 : [0, \infty] \quad \cdots \quad \Delta \vdash I_m : [0, \infty]\end{array}}{\Delta; \Gamma \vdash \mathtt{instance}(I_1, \ldots, I_m) \;\; E : \mathtt{Circuit}(\tau_1\{I_1/y_1, \ldots, I_m/y_m\}, \ldots, \tau_n\{I_1/y_1, \ldots, I_m/y_m\})}$$

$$\frac{\Delta; \Gamma \vdash U : \mathtt{Unit}}{\Delta; \Gamma \vdash \mathtt{reverse} \;\; U : \mathtt{Unit}} \qquad \frac{\begin{array}{c}\Delta \vdash I : [I_1, I_2] \qquad \Delta \vdash I' : [I_1', I_2'] \\ \Delta, y : [I, I']; \Gamma \vdash U : \mathtt{Unit}\end{array}}{\Delta; \Gamma \vdash \mathtt{for} \;\; y = I..I' \;\; \mathtt{do} \; \{ \; U \; \} : \mathtt{Unit}}$$

$$\frac{\begin{array}{c}\Delta, y_1 : [0, \infty], \ldots, y_m : [0, \infty] \vdash \tau_1 :: * \quad \cdots \quad \Delta, y_1 : [0, \infty], \ldots, y_m : [0, \infty] \vdash \tau_n :: * \\ \Delta, y_1 : [0, \infty], \ldots, y_m : [0, \infty]; \Gamma, x_1 : \tau_1, \ldots, x_n : \tau_n \vdash U : \mathtt{Unit}, \\ \Delta; \Gamma, x : \mathtt{Family}(y_1, \ldots, y_m)(\tau_1, \ldots, \tau_n) \vdash C : \mathtt{Unit}\end{array}}{\Delta; \Gamma \vdash \mathtt{family}(y_1, \ldots, y_m) \;\; x(x_1 : \tau_1, \ldots, x_n : \tau_n) \; \{ \; U \; \} \; \mathtt{in} \; \{ \; C \; \} : \mathtt{Unit}}$$

- Γ (type context) maps variables to types
- Judgements $\Delta \vdash \tau :: *$ assert that τ is "well typed" under index context Δ

Normalization

Theorem

If $\cdot;\cdot \vdash C : \texttt{Unit}$, *then*

$$\langle C, \emptyset, \lambda l.0, |00\cdots\rangle\rangle \Downarrow \langle \sigma, \eta, |\psi\rangle\rangle.$$

- Termination is easy
  - No recursion
  - No index variables **in evaluation contexts**
- Semi-difficult part is ruling out out-of-bounds access
  - Via $\Delta \models P$ judgments, substitution & inversion

Conclusion

In this talk...

- A typed dialect of QASM
  - Allows register & gates as parameters
  - Statically checks register accesses
- An extension of typedQASM supporting circuit family definitions & circuit reversal
  - Simple syntax, (relatively) simple & erasable type theory
  - Statically checks register accesses

Conclusion

In this talk...

- A typed dialect of QASM
  - Allows register & gates as parameters
  - Statically checks register accesses
- An extension of typedQASM supporting circuit family definitions & circuit reversal
  - Simple syntax, (relatively) simple & erasable type theory
  - Statically checks register accesses

Future work

- Implementation
- Decidability of type checking

Parametrized resource estimates

```
include "toffoli.qasm";
family(n) ctrlAdd(ctrl:Qbit, x:Qbit[n], y:Qbit[n], c:Qbit) {
    toffoli(x[0], ctrl, y[0]);
    cx(x[0], c);
    toffoli(c, y[0], x[0]);
    for i=1..n-2 do {
        toffoli(x[i], ctrl, y[i]);
        cx(x[i-1], x[i]);
        toffoli(x[i-1], y[i], x[i])
    }
    toffoli(x[n-1], ctrl, y[n-1]);
    toffoli(x[n-2], ctrl, y[n-1]);
    for i=2..n-1 do {
        toffoli(x[n-i-1], y[n-i], x[n-i]);
        cx(x[n-i-1], x[n-i]);
        toffoli(x[n-i-1], ctrl, y[n-i])
    }
    toffoli(c, y[0], x[0]);
    cx(x[0], c);
    toffoli(c, ctrl, y[0])
}
```

Parametrized resource estimates

```
include "toffoli.qasm";
family(n) ctrlAdd(ctrl:Qbit, x:Qbit[n], y:Qbit[n], c:Qbit) {
    toffoli(x[0], ctrl, y[0]);                    // 7
    cx(x[0], c);
    toffoli(c, y[0], x[0]);
    for i=1..n-2 do {
        toffoli(x[i], ctrl, y[i]);
        cx(x[i-1], x[i]);
        toffoli(x[i-1], y[i], x[i])
    }
    toffoli(x[n-1], ctrl, y[n-1]);
    toffoli(x[n-2], ctrl, y[n-1]);
    for i=2..n-1 do {
        toffoli(x[n-i-1], y[n-i], x[n-i]);
        cx(x[n-i-1], x[n-i]);
        toffoli(x[n-i-1], ctrl, y[n-i])
    }
    toffoli(c, y[0], x[0]);
    cx(x[0], c);
    toffoli(c, ctrl, y[0])
}
```

Parametrized resource estimates

```
include "toffoli.qasm";
family(n) ctrlAdd(ctrl:Qbit, x:Qbit[n], y:Qbit[n], c:Qbit) {
    toffoli(x[0], ctrl, y[0]);                    // 7
    cx(x[0], c);                                  // 0
    toffoli(c, y[0], x[0]);
    for i=1..n-2 do {
        toffoli(x[i], ctrl, y[i]);
        cx(x[i-1], x[i]);
        toffoli(x[i-1], y[i], x[i])
    }
    toffoli(x[n-1], ctrl, y[n-1]);
    toffoli(x[n-2], ctrl, y[n-1]);
    for i=2..n-1 do {
        toffoli(x[n-i-1], y[n-i], x[n-i]);
        cx(x[n-i-1], x[n-i]);
        toffoli(x[n-i-1], ctrl, y[n-i])
    }
    toffoli(c, y[0], x[0]);
    cx(x[0], c);
    toffoli(c, ctrl, y[0])
}
```

Parametrized resource estimates

```
include "toffoli.qasm";
family(n) ctrlAdd(ctrl:Qbit, x:Qbit[n], y:Qbit[n], c:Qbit) {
    toffoli(x[0], ctrl, y[0]);                     // 7
    cx(x[0], c);                                   // 0
    toffoli(c, y[0], x[0]);                        // 7
    for i=1..n-2 do {
        toffoli(x[i], ctrl, y[i]);
        cx(x[i-1], x[i]);
        toffoli(x[i-1], y[i], x[i])
    }
    toffoli(x[n-1], ctrl, y[n-1]);
    toffoli(x[n-2], ctrl, y[n-1]);
    for i=2..n-1 do {
        toffoli(x[n-i-1], y[n-i], x[n-i]);
        cx(x[n-i-1], x[n-i]);
        toffoli(x[n-i-1], ctrl, y[n-i])
    }
    toffoli(c, y[0], x[0]);
    cx(x[0], c);
    toffoli(c, ctrl, y[0])
}
```

Parametrized resource estimates

```
include "toffoli.qasm";
family(n) ctrlAdd(ctrl:Qbit, x:Qbit[n], y:Qbit[n], c:Qbit) {
    toffoli(x[0], ctrl, y[0]);                    // 7
    cx(x[0], c);                                  // 0
    toffoli(c, y[0], x[0]);                       // 7
    for i=1..n-2 do {                               // (n-2)*
        toffoli(x[i], ctrl, y[i]);
        cx(x[i-1], x[i]);
        toffoli(x[i-1], y[i], x[i])
    }
    toffoli(x[n-1], ctrl, y[n-1]);
    toffoli(x[n-2], ctrl, y[n-1]);
    for i=2..n-1 do {
        toffoli(x[n-i-1], y[n-i], x[n-i]);
        cx(x[n-i-1], x[n-i]);
        toffoli(x[n-i-1], ctrl, y[n-i])
    }
    toffoli(c, y[0], x[0]);
    cx(x[0], c);
    toffoli(c, ctrl, y[0])
}
```

Parametrized resource estimates

```
include "toffoli.qasm";
family(n) ctrlAdd(ctrl:Qbit, x:Qbit[n], y:Qbit[n], c:Qbit) {
    toffoli(x[0], ctrl, y[0]);                          // 7
    cx(x[0], c);                                        // 0
    toffoli(c, y[0], x[0]);                             // 7
    for i=1..n-2 do {                                     // (n-2)*
        toffoli(x[i], ctrl, y[i]);                        // 7
        cx(x[i-1], x[i]);
        toffoli(x[i-1], y[i], x[i])
    }
    toffoli(x[n-1], ctrl, y[n-1]);
    toffoli(x[n-2], ctrl, y[n-1]);
    for i=2..n-1 do {
        toffoli(x[n-i-1], y[n-i], x[n-i]);
        cx(x[n-i-1], x[n-i]);
        toffoli(x[n-i-1], ctrl, y[n-i])
    }
    toffoli(c, y[0], x[0]);
    cx(x[0], c);
    toffoli(c, ctrl, y[0])
}
```

Parametrized resource estimates

```
include "toffoli.qasm";
family(n) ctrlAdd(ctrl:Qbit, x:Qbit[n], y:Qbit[n], c:Qbit) {
    toffoli(x[0], ctrl, y[0]);                        // 7
    cx(x[0], c);                                      // 0
    toffoli(c, y[0], x[0]);                           // 7
    for i=1..n-2 do {                                   // (n-2)*
        toffoli(x[i], ctrl, y[i]);                      // 7
        cx(x[i-1], x[i]);                               // 0
        toffoli(x[i-1], y[i], x[i])
    }
    toffoli(x[n-1], ctrl, y[n-1]);
    toffoli(x[n-2], ctrl, y[n-1]);
    for i=2..n-1 do {
        toffoli(x[n-i-1], y[n-i], x[n-i]);
        cx(x[n-i-1], x[n-i]);
        toffoli(x[n-i-1], ctrl, y[n-i])
    }
    toffoli(c, y[0], x[0]);
    cx(x[0], c);
    toffoli(c, ctrl, y[0])
}
```

Parametrized resource estimates

```
include "toffoli.qasm";
family(n) ctrlAdd(ctrl:Qbit, x:Qbit[n], y:Qbit[n], c:Qbit) {
    toffoli(x[0], ctrl, y[0]);                      // 7
    cx(x[0], c);                                    // 0
    toffoli(c, y[0], x[0]);                         // 7
    for i=1..n-2 do {                                 // (n-2)*
        toffoli(x[i], ctrl, y[i]);                    // 7
        cx(x[i-1], x[i]);                             // 0
        toffoli(x[i-1], y[i], x[i])                   // 7
    }
    toffoli(x[n-1], ctrl, y[n-1]);
    toffoli(x[n-2], ctrl, y[n-1]);
    for i=2..n-1 do {
        toffoli(x[n-i-1], y[n-i], x[n-i]);
        cx(x[n-i-1], x[n-i]);
        toffoli(x[n-i-1], ctrl, y[n-i])
    }
    toffoli(c, y[0], x[0]);
    cx(x[0], c);
    toffoli(c, ctrl, y[0])
}
```

Parametrized resource estimates

```
include "toffoli.qasm";
family(n) ctrlAdd(ctrl:Qbit, x:Qbit[n], y:Qbit[n], c:Qbit) {
    toffoli(x[0], ctrl, y[0]);                              // 7
    cx(x[0], c);                                            // 0
    toffoli(c, y[0], x[0]);                                 // 7
    for i=1..n-2 do {                                         // (n-2)*
        toffoli(x[i], ctrl, y[i]);                            // 7
        cx(x[i-1], x[i]);                                     // 0
        toffoli(x[i-1], y[i], x[i])                           // 7
    }                                                       // 14(n-2)
    toffoli(x[n-1], ctrl, y[n-1]);
    toffoli(x[n-2], ctrl, y[n-1]);
    for i=2..n-1 do {
        toffoli(x[n-i-1], y[n-i], x[n-i]);
        cx(x[n-i-1], x[n-i]);
        toffoli(x[n-i-1], ctrl, y[n-i])
    }
    toffoli(c, y[0], x[0]);
    cx(x[0], c);
    toffoli(c, ctrl, y[0])
}
```

Parametrized resource estimates

```
include "toffoli.qasm";
family(n) ctrlAdd(ctrl:Qbit, x:Qbit[n], y:Qbit[n], c:Qbit) {
    toffoli(x[0], ctrl, y[0]);                      // 7
    cx(x[0], c);                                    // 0
    toffoli(c, y[0], x[0]);                         // 7
    for i=1..n-2 do {                                 // (n-2)*
        toffoli(x[i], ctrl, y[i]);                    // 7
        cx(x[i-1], x[i]);                             // 0
        toffoli(x[i-1], y[i], x[i])                   // 7
    }                                               // 14(n-2)
    toffoli(x[n-1], ctrl, y[n-1]);                  // 7
    toffoli(x[n-2], ctrl, y[n-1]);
    for i=2..n-1 do {
        toffoli(x[n-i-1], y[n-i], x[n-i]);
        cx(x[n-i-1], x[n-i]);
        toffoli(x[n-i-1], ctrl, y[n-i])
    }
    toffoli(c, y[0], x[0]);
    cx(x[0], c);
    toffoli(c, ctrl, y[0])
}
```

Parametrized resource estimates

```
include "toffoli.qasm";
family(n) ctrlAdd(ctrl:Qbit, x:Qbit[n], y:Qbit[n], c:Qbit) {
    toffoli(x[0], ctrl, y[0]);                     // 7
    cx(x[0], c);                                   // 0
    toffoli(c, y[0], x[0]);                        // 7
    for i=1..n-2 do {                                // (n-2)*
        toffoli(x[i], ctrl, y[i]);                   // 7
        cx(x[i-1], x[i]);                            // 0
        toffoli(x[i-1], y[i], x[i])                  // 7
    }                                              // 14(n-2)
    toffoli(x[n-1], ctrl, y[n-1]);                 // 7
    toffoli(x[n-2], ctrl, y[n-1]);                 // 7
    for i=2..n-1 do {
        toffoli(x[n-i-1], y[n-i], x[n-i]);
        cx(x[n-i-1], x[n-i]);
        toffoli(x[n-i-1], ctrl, y[n-i])
    }
    toffoli(c, y[0], x[0]);
    cx(x[0], c);
    toffoli(c, ctrl, y[0])
}
```

Parametrized resource estimates

```
include "toffoli.qasm";
family(n) ctrlAdd(ctrl:Qbit, x:Qbit[n], y:Qbit[n], c:Qbit) {
    toffoli(x[0], ctrl, y[0]);                       // 7
    cx(x[0], c);                                     // 0
    toffoli(c, y[0], x[0]);                          // 7
    for i=1..n-2 do {                                  // (n-2)*
        toffoli(x[i], ctrl, y[i]);                     // 7
        cx(x[i-1], x[i]);                              // 0
        toffoli(x[i-1], y[i], x[i])                    // 7
    }                                                // 14(n-2)
    toffoli(x[n-1], ctrl, y[n-1]);                   // 7
    toffoli(x[n-2], ctrl, y[n-1]);                   // 7
    for i=2..n-1 do {                                  // (n-2)*
        toffoli(x[n-i-1], y[n-i], x[n-i]);
        cx(x[n-i-1], x[n-i]);
        toffoli(x[n-i-1], ctrl, y[n-i])
    }
    toffoli(c, y[0], x[0]);
    cx(x[0], c);
    toffoli(c, ctrl, y[0])
}
```

Parametrized resource estimates

```
include "toffoli.qasm";
family(n) ctrlAdd(ctrl:Qbit, x:Qbit[n], y:Qbit[n], c:Qbit) {
    toffoli(x[0], ctrl, y[0]);                        // 7
    cx(x[0], c);                                      // 0
    toffoli(c, y[0], x[0]);                           // 7
    for i=1..n-2 do {                                   // (n-2)*
        toffoli(x[i], ctrl, y[i]);                      // 7
        cx(x[i-1], x[i]);                               // 0
        toffoli(x[i-1], y[i], x[i])                     // 7
    }                                                 // 14(n-2)
    toffoli(x[n-1], ctrl, y[n-1]);                    // 7
    toffoli(x[n-2], ctrl, y[n-1]);                    // 7
    for i=2..n-1 do {                                   // (n-2)*
        toffoli(x[n-i-1], y[n-i], x[n-i]);              // 7
        cx(x[n-i-1], x[n-i]);
        toffoli(x[n-i-1], ctrl, y[n-i])
    }
    toffoli(c, y[0], x[0]);
    cx(x[0], c);
    toffoli(c, ctrl, y[0])
}
```

Parametrized resource estimates

```
include "toffoli.qasm";
family(n) ctrlAdd(ctrl:Qbit, x:Qbit[n], y:Qbit[n], c:Qbit) {
    toffoli(x[0], ctrl, y[0]);                    // 7
    cx(x[0], c);                                  // 0
    toffoli(c, y[0], x[0]);                       // 7
    for i=1..n-2 do {                               // (n-2)*
        toffoli(x[i], ctrl, y[i]);                  // 7
        cx(x[i-1], x[i]);                           // 0
        toffoli(x[i-1], y[i], x[i])                 // 7
    }                                             // 14(n-2)
    toffoli(x[n-1], ctrl, y[n-1]);                // 7
    toffoli(x[n-2], ctrl, y[n-1]);                // 7
    for i=2..n-1 do {                               // (n-2)*
        toffoli(x[n-i-1], y[n-i], x[n-i]);          // 7
        cx(x[n-i-1], x[n-i]);                       // 0
        toffoli(x[n-i-1], ctrl, y[n-i])
    }
    toffoli(c, y[0], x[0]);
    cx(x[0], c);
    toffoli(c, ctrl, y[0])
}
```

Parametrized resource estimates

```
include "toffoli.qasm";
family(n) ctrlAdd(ctrl:Qbit, x:Qbit[n], y:Qbit[n], c:Qbit) {
    toffoli(x[0], ctrl, y[0]);                          // 7
    cx(x[0], c);                                        // 0
    toffoli(c, y[0], x[0]);                             // 7
    for i=1..n-2 do {                                     // (n-2)*
        toffoli(x[i], ctrl, y[i]);                        // 7
        cx(x[i-1], x[i]);                                 // 0
        toffoli(x[i-1], y[i], x[i])                       // 7
    }                                                   // 14(n-2)
    toffoli(x[n-1], ctrl, y[n-1]);                      // 7
    toffoli(x[n-2], ctrl, y[n-1]);                      // 7
    for i=2..n-1 do {                                     // (n-2)*
        toffoli(x[n-i-1], y[n-i], x[n-i]);                // 7
        cx(x[n-i-1], x[n-i]);                             // 0
        toffoli(x[n-i-1], ctrl, y[n-i])                   // 7
    }
    toffoli(c, y[0], x[0]);
    cx(x[0], c);
    toffoli(c, ctrl, y[0])
}
```

Parametrized resource estimates

```
include "toffoli.qasm";
family(n) ctrlAdd(ctrl:Qbit, x:Qbit[n], y:Qbit[n], c:Qbit) {
    toffoli(x[0], ctrl, y[0]);                  // 7
    cx(x[0], c);                                // 0
    toffoli(c, y[0], x[0]);                     // 7
    for i=1..n-2 do {                             // (n-2)*
        toffoli(x[i], ctrl, y[i]);                // 7
        cx(x[i-1], x[i]);                         // 0
        toffoli(x[i-1], y[i], x[i])               // 7
    }                                           // 14(n-2)
    toffoli(x[n-1], ctrl, y[n-1]);              // 7
    toffoli(x[n-2], ctrl, y[n-1]);              // 7
    for i=2..n-1 do {                             // (n-2)*
        toffoli(x[n-i-1], y[n-i], x[n-i]);        // 7
        cx(x[n-i-1], x[n-i]);                     // 0
        toffoli(x[n-i-1], ctrl, y[n-i])           // 7
    }                                           // 14(n-2)
    toffoli(c, y[0], x[0]);
    cx(x[0], c);
    toffoli(c, ctrl, y[0])
}
```

Parametrized resource estimates

```
include "toffoli.qasm";
family(n) ctrlAdd(ctrl:Qbit, x:Qbit[n], y:Qbit[n], c:Qbit) {
    toffoli(x[0], ctrl, y[0]);                        // 7
    cx(x[0], c);                                      // 0
    toffoli(c, y[0], x[0]);                           // 7
    for i=1..n-2 do {                                   // (n-2)*
        toffoli(x[i], ctrl, y[i]);                      // 7
        cx(x[i-1], x[i]);                               // 0
        toffoli(x[i-1], y[i], x[i])                     // 7
    }                                                 // 14(n-2)
    toffoli(x[n-1], ctrl, y[n-1]);                    // 7
    toffoli(x[n-2], ctrl, y[n-1]);                    // 7
    for i=2..n-1 do {                                   // (n-2)*
        toffoli(x[n-i-1], y[n-i], x[n-i]);              // 7
        cx(x[n-i-1], x[n-i]);                           // 0
        toffoli(x[n-i-1], ctrl, y[n-i])                 // 7
    }                                                 // 14(n-2)
    toffoli(c, y[0], x[0]);                           // 7
    cx(x[0], c);
    toffoli(c, ctrl, y[0])
}
```

Parametrized resource estimates

```
include "toffoli.qasm";
family(n) ctrlAdd(ctrl:Qbit, x:Qbit[n], y:Qbit[n], c:Qbit) {
    toffoli(x[0], ctrl, y[0]);                    // 7
    cx(x[0], c);                                  // 0
    toffoli(c, y[0], x[0]);                       // 7
    for i=1..n-2 do {                               // (n-2)*
        toffoli(x[i], ctrl, y[i]);                  // 7
        cx(x[i-1], x[i]);                           // 0
        toffoli(x[i-1], y[i], x[i])                 // 7
    }                                             // 14(n-2)
    toffoli(x[n-1], ctrl, y[n-1]);                // 7
    toffoli(x[n-2], ctrl, y[n-1]);                // 7
    for i=2..n-1 do {                               // (n-2)*
        toffoli(x[n-i-1], y[n-i], x[n-i]);          // 7
        cx(x[n-i-1], x[n-i]);                       // 0
        toffoli(x[n-i-1], ctrl, y[n-i])             // 7
    }                                             // 14(n-2)
    toffoli(c, y[0], x[0]);                       // 7
    cx(x[0], c);                                  // 0
    toffoli(c, ctrl, y[0])
}
```

Parametrized resource estimates

```
include "toffoli.qasm";
family(n) ctrlAdd(ctrl:Qbit, x:Qbit[n], y:Qbit[n], c:Qbit) {
    toffoli(x[0], ctrl, y[0]);                    // 7
    cx(x[0], c);                                  // 0
    toffoli(c, y[0], x[0]);                       // 7
    for i=1..n-2 do {                               // (n-2)*
        toffoli(x[i], ctrl, y[i]);                  // 7
        cx(x[i-1], x[i]);                           // 0
        toffoli(x[i-1], y[i], x[i])                 // 7
    }                                             // 14(n-2)
    toffoli(x[n-1], ctrl, y[n-1]);                // 7
    toffoli(x[n-2], ctrl, y[n-1]);                // 7
    for i=2..n-1 do {                               // (n-2)*
        toffoli(x[n-i-1], y[n-i], x[n-i]);          // 7
        cx(x[n-i-1], x[n-i]);                       // 0
        toffoli(x[n-i-1], ctrl, y[n-i])             // 7
    }                                             // 14(n-2)
    toffoli(c, y[0], x[0]);                       // 7
    cx(x[0], c);                                  // 0
    toffoli(c, ctrl, y[0])                        // 7
}
```

Parametrized resource estimates

```
include "toffoli.qasm";
family(n) ctrlAdd(ctrl:Qbit, x:Qbit[n], y:Qbit[n], c:Qbit) {
    toffoli(x[0], ctrl, y[0]);                    // 7
    cx(x[0], c);                                  // 0
    toffoli(c, y[0], x[0]);                       // 7
    for i=1..n-2 do {                               // (n-2)*
        toffoli(x[i], ctrl, y[i]);                  // 7
        cx(x[i-1], x[i]);                           // 0
        toffoli(x[i-1], y[i], x[i])                 // 7
    }                                             // 14(n-2)
    toffoli(x[n-1], ctrl, y[n-1]);                // 7
    toffoli(x[n-2], ctrl, y[n-1]);                // 7
    for i=2..n-1 do {                               // (n-2)*
        toffoli(x[n-i-1], y[n-i], x[n-i]);          // 7
        cx(x[n-i-1], x[n-i]);                       // 0
        toffoli(x[n-i-1], ctrl, y[n-i])             // 7
    }                                             // 14(n-2)
    toffoli(c, y[0], x[0]);                       // 7
    cx(x[0], c);                                  // 0
    toffoli(c, ctrl, y[0])                        // 7
}                                               // 28n - 14
```

Optimization of circuit families

```
include "toffoli.qasm";
family(n) ctrlAdd(ctrl:Qbit, x:Qbit[n], y:Qbit[n], c:Qbit) {
    toffoli(x[0], ctrl, y[0]);                  // 7
    cx(x[0], c);                                // 0
    toffoli(c, y[0], x[0]);                     // 7
    for i=1..n-2 do {                             // (n-2)*
        toffoli(x[i], ctrl, y[i]);                // 7
        cx(x[i-1], x[i]);                         // 0
        toffoli(x[i-1], y[i], x[i])               // 7
    }                                           // 14(n-2)
    toffoli(x[n-1], ctrl, y[n-1]);              // 7
    toffoli(x[n-2], ctrl, y[n-1]);              // 7
    for i=2..n-1 do {                             // (n-2)*
        toffoli(x[n-i-1], y[n-i], x[n-i]);        // 7
        cx(x[n-i-1], x[n-i]);                     // 0
        toffoli(x[n-i-1], ctrl, y[n-i])           // 7
    }                                           // 14(n-2)
    toffoli(c, y[0], x[0]);                     // 7
    cx(x[0], c);                                // 0
    toffoli(c, ctrl, y[0])                      // 7
}                                             // 28n - 14
```

Optimization of circuit families

```
include "toffoli.qasm";
family(n) ctrlAdd(ctrl:Qbit, x:Qbit[n], y:Qbit[n], c:Qbit) {
    toffoli(x[0], ctrl, y[0]);                       // 7
    cx(x[0], c);                                     // 0
    toffoli(c, y[0], x[0]);                          // 7
    for i=1..n-2 do {                                  // (n-2)*
        toffoli(x[i], ctrl, y[i]);                     // 7
        cx(x[i-1], x[i]);                              // 0
        toffoli(x[i-1], y[i], x[i])                    // 7
    }                                                // 14(n-2)
    tof_opt(x[n-1], ctrl, y[n-1]);                  // 4
    tof_opt(x[n-2], ctrl, y[n-1]);                  // 4
    for i=2..n-1 do {                                  // (n-2)*
        toffoli(x[n-i-1], y[n-i], x[n-i]);             // 7
        cx(x[n-i-1], x[n-i]);                          // 0
        toffoli(x[n-i-1], ctrl, y[n-i])                // 7
    }                                                // 14(n-2)
    toffoli(c, y[0], x[0]);                          // 7
    cx(x[0], c);                                     // 0
    toffoli(c, ctrl, y[0])                           // 7
}                                                 // 28n - 8
```

Optimization of circuit families

```
include "toffoli.qasm";
family(n) ctrlAdd(ctrl:Qbit, x:Qbit[n], y:Qbit[n], c:Qbit) {
    toffoli(x[0], ctrl, y[0]);                          // 7
    cx(x[0], c);                                        // 0
    toffoli(c, y[0], x[0]);                             // 7
    for i=1..n-2 do {                                     // (n-2)*
        toffoli(x[i], ctrl, y[i]);                        // 7
        cx(x[i-1], x[i]);                                 // 0
        toffoli(x[i-1], y[i], x[i])                       // 7
    }                                                   // 14(n-2)
    tof_opt(x[n-1], ctrl, y[n-1]);                     // 4
    tof_opt(x[n-2], ctrl, y[n-1]);                     // 4
    for i=2..n-1 do {                                     // (n-2)*
        tof_opt(x[n-i-1], y[n-i], x[n-i]);               // 6
        cx(x[n-i-1], x[n-i]);                            // 0
        tof_opt(x[n-i-1], ctrl, y[n-i])                  // 6
    }                                                   // 12(n-2)
    toffoli(c, y[0], x[0]);                             // 7
    cx(x[0], c);                                        // 0
    toffoli(c, ctrl, y[0])                              // 7
}                                                      // 26n - 10
```

Optimization of circuit families

```
include "toffoli.qasm";
family(n) ctrlAdd(ctrl:Qbit, x:Qbit[n], y:Qbit[n], c:Qbit) {
    toffoli(x[0], ctrl, y[0]);                          // 7
    cx(x[0], c);                                        // 0
    tof_opt(c, y[0], x[0]);                             // 4
    for i=1..n-2 do {                                     // (n-2)*
        toffoli(x[i], ctrl, y[i]);                        // 7
        cx(x[i-1], x[i]);                                 // 0
        toffoli(x[i-1], y[i], x[i])                       // 7
    }                                                   // 14(n-2)
    tof_opt(x[n-1], ctrl, y[n-1]);                     // 4
    tof_opt(x[n-2], ctrl, y[n-1]);                     // 4
    for i=2..n-1 do {                                     // (n-2)*
        tof_opt(x[n-i-1], y[n-i], x[n-i]);               // 6
        cx(x[n-i-1], x[n-i]);                             // 0
        tof_opt(x[n-i-1], ctrl, y[n-i])                  // 6
    }                                                   // 12(n-2)
    tof_opt(c, y[0], x[0]);                             // 4
    cx(x[0], c);                                        // 0
    toffoli(c, ctrl, y[0])                              // 7
}                                                     // 26n - 16
```

Thank you!