Resource-Aware Hybrid Quantum Programming with General Recursion and Quantum Control

Kostia Chardonnet, Emmanuel Hainry, Romain Péchoux, Thomas Vinet

15/01/2026

LORIA, Nancy

Talk outline

Motivations

`Hyrql`: a hybrid language

Term rewrite systems and resource analysis

Future work

1. Motivations

Motivations

History of quantum algorithms:

- Based on the QRAM model (Shor): classical control

Motivations

History of quantum algorithms:

- Based on the QRAM model (Shor): classical control
- Then, interest in quantum control:

$$\mathtt{QS}(U, V) = |0\rangle\langle 0| \otimes UV + |1\rangle\langle 1| \otimes VU$$

Motivations

History of quantum algorithms:

- Based on the QRAM model (Shor): classical control
- Then, interest in quantum control:

$$\mathtt{QS}(U, V) = |0\rangle\langle 0| \otimes UV + |1\rangle\langle 1| \otimes VU$$

- Next steps: build a *hybrid language*, i.e. with both control flows

Contributions

`Hyrql`: a hybrid language with general recursion and classical control

Contributions

`Hyrql`: a hybrid language with general recursion and classical control

- Extension of Symmetric-Pattern Matching [SVV18]

Contributions

`Hyrql`: a hybrid language with general recursion and classical control

- Extension of Symmetric-Pattern Matching [SVV18]
- Typing discipline to ensure feasibility

Contributions

`Hyrql`: a hybrid language with general recursion and classical control

- Extension of Symmetric-Pattern Matching [SVV18]
- Typing discipline to ensure feasibility
- No measurement, but strictly more that a Circuit Description Language

Contributions

`Hyrql`: a hybrid language with general recursion and classical control

- Extension of Symmetric-Pattern Matching [SVV18]
- Typing discipline to ensure feasibility
- No measurement, but strictly more that a Circuit Description Language
- Polynomial termination ensures polynomial size circuits

Contributions

`Hyrql`: a hybrid language with general recursion and classical control

- Extension of Symmetric-Pattern Matching [SVV18]
- Typing discipline to ensure feasibility
- No measurement, but strictly more that a Circuit Description Language
- Polynomial termination ensures polynomial size circuits
- Amenable to static analysis through a translation to Term Rewrite Systems

2. `Hyrql`: a hybrid language

Syntax

$$t ::= x \mid \lambda x.t \mid t_1 t_2$$

Syntax

$$t ::= x \mid \lambda x.t \mid t_1\, t_2$$
$$\mid c(t_1, \ldots, t_n) \mid \texttt{match}\, t \left\{ c_1(\overrightarrow{x_1}) \to t_1 \,, \ldots, c_n(\overrightarrow{x_n}) \to t_n \right\}$$

- Example of standard constructs: unit $()$, pairs $a \otimes b$, numbers $0, S(n)$, lists $[\,], h :: t$

Syntax

$$
\begin{aligned}
t ::= {} & x \mid \lambda x.t \mid t_1\, t_2 \\
& \mid c(t_1, \ldots, t_n) \mid \mathtt{match}\, t \left\{ c_1(\overrightarrow{x_1}) \to t_1 \,, \ldots, c_n(\overrightarrow{x_n}) \to t_n \right\} \\
& \mid |0\rangle \mid |1\rangle \mid \mathtt{qcase}\, t \left\{ |0\rangle \to t_0 \,, |1\rangle \to t_1 \right\}
\end{aligned}
$$

- Example of standard constructs: unit $()$, pairs $a \otimes b$, numbers $0, S(n)$, lists $[\,], h :: t$

Syntax

$$
\begin{aligned}
t ::= {} & x \mid \lambda x.t \mid t_1\, t_2 \\
& \mid c(t_1, \ldots, t_n) \mid \mathtt{match}\, t \left\{ c_1(\overrightarrow{x_1}) \to t_1 \,, \ldots, c_n(\overrightarrow{x_n}) \to t_n \right\} \\
& \mid |0\rangle \mid |1\rangle \mid \mathtt{qcase}\, t \left\{ |0\rangle \to t_0 \,, |1\rangle \to t_1 \right\} \\
& \mid \mathtt{letrec}\, f\, x = t
\end{aligned}
$$

- Example of standard constructs: unit $()$, pairs $a \otimes b$, numbers $0, S(n)$, lists $[\,], h :: t$

Syntax

$$
\begin{aligned}
t ::= {} & x \mid \lambda x.t \mid t_1\, t_2 \\
& \mid c(t_1, \ldots, t_n) \mid \texttt{match}\, t \left\{ c_1(\overrightarrow{x_1}) \to t_1 \,, \ldots, c_n(\overrightarrow{x_n}) \to t_n \right\} \\
& \mid |0\rangle \mid |1\rangle \mid \texttt{qcase}\, t \left\{ |0\rangle \to t_0 \,, |1\rangle \to t_1 \right\} \\
& \mid \texttt{letrec}\, f\, x = t \mid \textstyle\sum_{i=1}^{n} \alpha_i \cdot t_i
\end{aligned}
$$

- Example of standard constructs: unit $()$, pairs $a \otimes b$, numbers $0, S(n)$, lists $[\,], h :: t$

Syntax

$$
\begin{aligned}
t ::= {} & x \mid \lambda x.t \mid t_1\, t_2 \\
& \mid c(t_1, \ldots, t_n) \mid \mathtt{match}\, t \left\{ c_1(\overrightarrow{x_1}) \to t_1 \,, \ldots, c_n(\overrightarrow{x_n}) \to t_n \right\} \\
& \mid |0\rangle \mid |1\rangle \mid \mathtt{qcase}\, t \left\{ |0\rangle \to t_0 \,, |1\rangle \to t_1 \right\} \\
& \mid \mathtt{letrec}\, f\, x = t \mid \textstyle\sum_{i=1}^{n} \alpha_i \cdot t_i \mid \mathtt{shape}(t)
\end{aligned}
$$

- Example of standard constructs: unit $()$, pairs $a \otimes b$, numbers $0, S(n)$, lists $[\,], h :: t$
- `shape` extracts the classical structure:
  $|0\rangle :: |+\rangle :: [\,] \to () :: () :: [\,]$

Simple quantum gate example

Standard gate: the NOT gate

$$\texttt{NOT} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Simple quantum gate example

Standard gate: the NOT gate

$$\texttt{NOT} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

In `Hyrql`:

$$\texttt{NOT} = \lambda x.\texttt{qcase}\, x \begin{Bmatrix} |0\rangle \to |1\rangle \\ |1\rangle \to |0\rangle \end{Bmatrix}$$

Abstraction

Simple quantum gate example

Standard gate: the NOT gate

$$\texttt{NOT} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

In `Hyrql`:

$$\texttt{NOT} = \lambda x.\texttt{qcase}\, x \left\{ \begin{array}{l} |0\rangle \to |1\rangle \\ |1\rangle \to |0\rangle \end{array} \right\}$$

Pattern-matching with two branches

Simple quantum gate example

Standard gate: the NOT gate

$$\texttt{NOT} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

In `Hyrql`:

$$\texttt{NOT} = \lambda x.\texttt{qcase}\, x \begin{Bmatrix} |0\rangle \to |1\rangle \\ |1\rangle \to |0\rangle \end{Bmatrix}$$

Output branches

Other examples

$$\texttt{cnot} \triangleq \lambda c.\lambda t.\texttt{qcase}\, c \left\{ |0\rangle \to |0\rangle \otimes t\,, |1\rangle \to |1\rangle \otimes \texttt{not}\, t \right\}$$

Other examples

$$\texttt{cnot} \triangleq \lambda c.\lambda t.\texttt{qcase}\, c \left\{ |0\rangle \to |0\rangle \otimes t\,, |1\rangle \to |1\rangle \otimes \texttt{not}\, t \right\}$$

$$\texttt{len} \triangleq \texttt{letrec}\, f\, l = \texttt{match}\, l \left\{ [\,] \to 0, h :: t \to S(f\ t) \right\}$$

Other examples

$$\texttt{cnot} \triangleq \lambda c.\lambda t.\texttt{qcase}\, c \left\{ |0\rangle \to |0\rangle \otimes t\,, |1\rangle \to |1\rangle \otimes \texttt{not}\, t \right\}$$

$$\texttt{len} \triangleq \texttt{letrec}\, f\, l = \texttt{match}\, l \left\{ [\,] \to 0, h :: t \to S(f\ t) \right\}$$

$$\texttt{repeat} \triangleq \texttt{letrec}\, f\, n = \texttt{match}\, n \left\{ 0 \to [\,], S(a) \to |0\rangle :: (f\ a) \right\}$$

$$\texttt{bqwalk} \triangleq$$

$$\texttt{letrec}\, f\, q = \lambda n.\texttt{qcase}\, q \left\{ \begin{array}{l} |0\rangle \to |0\rangle :: (\texttt{repeat}\ n) \\ |1\rangle \to \texttt{match}\, n \left\{ \begin{array}{c} 0 \to |1\rangle :: [\,] \\ S(m) \to |1\rangle :: (f\, |+\rangle\, m) \end{array} \right\} \end{array} \right\}$$

Operational semantics

- Follows a *Call-by-value* strategy

Operational semantics

- Follows a *Call-by-value* strategy
- $(\lambda x.t)v \rightsquigarrow t\{v/x\}$

Operational semantics

- Follows a *Call-by-value* strategy
- $(\lambda x.t)v \rightsquigarrow t\{v/x\}$
- $\texttt{qcase}\ |0\rangle \left\{ |0\rangle \to t_0\,, |1\rangle \to t_1 \right\} \rightsquigarrow t_0$

Operational semantics

- Follows a *Call-by-value* strategy
- $(\lambda x.t)v \rightsquigarrow t\{v/x\}$
- $\texttt{qcase}\ |0\rangle \left\{ |0\rangle \to t_0\,, |1\rangle \to t_1 \right\} \rightsquigarrow t_0$
- $\texttt{qcase}\ |1\rangle \left\{ |0\rangle \to t_0\,, |1\rangle \to t_1 \right\} \rightsquigarrow t_1$

Operational semantics

- Follows a *Call-by-value* strategy
- $(\lambda x.t)v \rightsquigarrow t\{v/x\}$
- $\texttt{qcase}\ |0\rangle \left\{ |0\rangle \to t_0\,, |1\rangle \to t_1 \right\} \rightsquigarrow t_0$
- $\texttt{qcase}\ |1\rangle \left\{ |0\rangle \to t_0\,, |1\rangle \to t_1 \right\} \rightsquigarrow t_1$

$$\texttt{qcase}\ \alpha \cdot |0\rangle + \beta \cdot |1\rangle \left\{ |0\rangle \to t_0\,, |1\rangle \to t_1 \right\} \rightsquigarrow \alpha \cdot t_0 + \beta \cdot t_1$$

Operational semantics

- Follows a *Call-by-value* strategy
- $(\lambda x.t)v \rightsquigarrow t\{v/x\}$
- $\texttt{qcase}\ |0\rangle \left\{ |0\rangle \to t_0\,, |1\rangle \to t_1 \right\} \rightsquigarrow t_0$
- $\texttt{qcase}\ |1\rangle \left\{ |0\rangle \to t_0\,, |1\rangle \to t_1 \right\} \rightsquigarrow t_1$

$$\texttt{qcase}\ \alpha \cdot |0\rangle + \beta \cdot |1\rangle \left\{ |0\rangle \to t_0\,, |1\rangle \to t_1 \right\} \rightsquigarrow \alpha \cdot t_0 + \beta \cdot t_1$$

- Superposition: parallel reduction

Type system

$$T ::= \mathrm{Qbit} \mid B \mid T \multimap T \mid T \Rightarrow T$$

- Constructed types $B : \texttt{nat}, A \otimes B, \texttt{list}(B)$
- Both linear and non-linear
- Typing judgment $\Gamma; \Delta \vdash t : T$, with a linear context Δ and a non-linear context Γ
- Orthogonality predicate $s \perp t$: same classical structure and $\langle s, t \rangle = 0$

Some typing rules

$$\frac{}{\Gamma, x : C; \varnothing \vdash x : C} \qquad \frac{}{\Gamma; x : Q \vdash x : Q}$$

$$\frac{\Gamma, x : C; \Delta \vdash t : T}{\Gamma; \Delta \vdash \lambda x.t : C \Rightarrow T} \qquad \frac{\Gamma; \Delta, x : Q \vdash t : T'}{\Gamma; \Delta \vdash \lambda x.t : Q \multimap T'}$$

$$\frac{\Gamma; \Delta \vdash t_i : Q \qquad \sum_{i=1}^{n} |\alpha_i|^2 = 1 \qquad \forall i \neq j,\ t_i \perp t_j}{\Gamma; \Delta \vdash \sum_{i=1}^{n} \alpha_i \cdot t_i : Q}$$

Some typing rules

$$\frac{}{\Gamma, x : C; \varnothing \vdash x : C} \qquad \frac{}{\Gamma; x : Q \vdash x : Q}$$

$$\frac{\Gamma, x : C; \Delta \vdash t : T}{\Gamma; \Delta \vdash \lambda x.t : C \Rightarrow T} \qquad \frac{\Gamma; \Delta, x : Q \vdash t : T'}{\Gamma; \Delta \vdash \lambda x.t : Q \multimap T'}$$

$$\frac{\Gamma; \Delta \vdash t_i : Q \qquad \sum_{i=1}^{n} |\alpha_i|^2 = 1 \qquad \forall i \neq j,\ t_i \perp t_j}{\Gamma; \Delta \vdash \sum_{i=1}^{n} \alpha_i \cdot t_i : Q}$$

Back on $\texttt{len} \triangleq \texttt{letrec}\, f\ l = \texttt{match}\, l \left\{ [\,] \to 0, h :: t \to S(f\ t) \right\}$:

- $\not\vdash \texttt{len} : [Q] \multimap \texttt{nat}$ but $\vdash \texttt{len} : [C] \multimap \texttt{nat}$;
- Using shape: $\vdash \lambda x.(x \otimes \texttt{len}(\texttt{shape}(x))) : [Q] \multimap [Q] \otimes \texttt{nat}$

Properties (1/2)

- Confluence and progress are verified
- Subject reduction holds only for terminating terms or classical terms
- Orthogonality is Π_2^0-complete, but can be decided polynomially in most cases

Properties (2/2)

Circuit compilation

Let t be a term of $\texttt{Hyrql}_r$, and v be an input. If tv in time $\mathrm{Poly}(|v|)$, then there exists a circuit $\mathcal{C}$ of size $\mathrm{Poly}(|v|)$ computing tv.

Termination and complexity certificate implies certificate on the circuit

3. Term rewrite systems and resource analysis

Term rewrite systems

- Represent a program as a set of rules $\mathcal{R}$

$$\texttt{Not}\ |0\rangle \to |1\rangle \qquad \texttt{Not}\ |1\rangle \to |0\rangle$$

- STTRS: extension for higher-order [Yam01]
- Active field for both termination and complexity analysis: different existing techniques
- **Compile** `Hyrql` into TRS

Translation idea

- Produce $\mathcal{R}$ inductively, producing rules for each branch of `qcase` / `match`
- Keep track of what branch was taken
- When t is closed and higher-order: associate $\mathtt{f}_t$

Translation idea

- Produce $\mathcal{R}$ inductively, producing rules for each branch of `qcase` / `match`
- Keep track of what branch was taken
- When t is closed and higher-order: associate $\mathtt{f}_t$

$$\lambda x.\lambda y.\mathtt{qcase}\, x \left\{ |0\rangle \to |0\rangle \otimes y \,,\, |1\rangle \to |1\rangle \otimes \mathtt{Not}\, y \right\}$$

Translation idea

- Produce $\mathcal{R}$ inductively, producing rules for each branch of `qcase` / `match`
- Keep track of what branch was taken
- When t is closed and higher-order: associate $\mathtt{f}_t$

$$\lambda x.\lambda y.\mathtt{qcase}\, x \left\{ |0\rangle \to |0\rangle \otimes y \,,\, |1\rangle \to |1\rangle \otimes \mathtt{Not}\, y \right\}$$

$$\mathcal{R} = \{\bot \to |0\rangle \otimes y\}$$

Translation idea

- Produce $\mathcal{R}$ inductively, producing rules for each branch of `qcase` / `match`
- Keep track of what branch was taken
- When t is closed and higher-order: associate $\mathtt{f}_t$

$$\lambda x.\lambda y.\mathtt{qcase}\, x \left\{ |0\rangle \to |0\rangle \otimes y \,,\, |1\rangle \to |1\rangle \otimes \mathtt{Not}\, y \right\}$$

$$\mathcal{R} = \{\bot \to |1\rangle \otimes \mathtt{Not}\, y\}$$

Translation idea

- Produce $\mathcal{R}$ inductively, producing rules for each branch of `qcase` / `match`
- Keep track of what branch was taken
- When t is closed and higher-order: associate $\mathtt{f}_t$

$$\lambda x.\lambda y.\mathtt{qcase}\, x \left\{ |0\rangle \to |0\rangle \otimes y \,, |1\rangle \to |1\rangle \otimes \mathtt{Not}\, y \right\}$$

$$\mathcal{R} = \{\bot \to |0\rangle \otimes y : |0\rangle / x, \bot \to |1\rangle \otimes \mathtt{Not}\, y : |1\rangle / x\}$$

Translation idea

- Produce $\mathcal{R}$ inductively, producing rules for each branch of `qcase` / `match`
- Keep track of what branch was taken
- When t is closed and higher-order: associate $\mathtt{f}_t$

$$\lambda x.\lambda y.\mathtt{qcase}\, x \left\{ |0\rangle \to |0\rangle \otimes y \,, |1\rangle \to |1\rangle \otimes \mathtt{Not}\, y \right\}$$

$$\mathcal{R} = \{ y \to |0\rangle \otimes y : |0\rangle / x, y \to |1\rangle \otimes \mathtt{Not}\, y : |1\rangle / x \}$$

Translation idea

- Produce $\mathcal{R}$ inductively, producing rules for each branch of `qcase` / `match`
- Keep track of what branch was taken
- When t is closed and higher-order: associate $\mathtt{f}_t$

$$\lambda x.\lambda y.\mathtt{qcase}\, x \left\{ |0\rangle \to |0\rangle \otimes y \,,\, |1\rangle \to |1\rangle \otimes \mathtt{Not}\, y \right\}$$

$$\mathcal{R} = \{ |0\rangle\ y \to |0\rangle \otimes y, |1\rangle\ y \to |1\rangle \otimes \mathtt{Not}\, y \}$$

Translation idea

- Produce $\mathcal{R}$ inductively, producing rules for each branch of `qcase` / `match`
- Keep track of what branch was taken
- When t is closed and higher-order: associate $\mathtt{f}_t$

$$\lambda x.\lambda y.\mathtt{qcase}\, x \left\{ |0\rangle \to |0\rangle \otimes y\,, |1\rangle \to |1\rangle \otimes \mathtt{Not}\, y \right\}$$

$$\mathcal{R} = \{\mathtt{CNot}\ |0\rangle\ y \to |0\rangle \otimes y, \mathtt{CNot}\ |1\rangle\ y \to |1\rangle \otimes \mathtt{Not}\, y\}$$

Translation idea

- Produce $\mathcal{R}$ inductively, producing rules for each branch of `qcase` / `match`
- Keep track of what branch was taken
- When t is closed and higher-order: associate $\mathtt{f}_t$

$$\lambda x.\lambda y.\mathtt{qcase}\, x \left\{ |0\rangle \to |0\rangle \otimes y \,, |1\rangle \to |1\rangle \otimes \mathtt{Not}\, y \right\}$$

$$\mathcal{R} = \{\mathtt{CNot}\ |0\rangle\ y \to |0\rangle \otimes y, \mathtt{CNot}\ |1\rangle\ y \to |1\rangle \otimes \mathtt{Not}\, y\}$$

- Restrict the syntax, but $\mathcal{O}(n) \to \mathcal{O}(n^2)$

Translation properties

Translation well-definedness

Let $\mathcal{R} = \texttt{Translate}(t)$. Then $\mathcal{R}$ is a well-defined STTRS, and is computed in $\mathcal{O}(|t|^4)$.

Translation properties

Translation well-definedness

Let $\mathcal{R} = \texttt{Translate}(t)$. Then $\mathcal{R}$ is a well-defined STTRS, and is computed in $\mathcal{O}(|t|^4)$.

Semantics and complexity preservation

Let $\mathcal{R} = \texttt{Translate}(t)$. Then, $t, \mathcal{R}$ terminate on the same inputs, to the same output. When they terminate in k, l steps, $k = \Theta(l)$.

Coming back on our example

`Translate(bqwalk)` yields the following system:

$$\left\{\begin{array}{rr} \texttt{Repeat}\,0 \to [\,] & \texttt{Repeat}\,S(n) \to |0\rangle :: \texttt{Repeat}\,n \\ \texttt{BQWalk}\,|1\rangle\,0 \to |1\rangle :: [\,] & \texttt{BQWalk}\,|0\rangle\,n \to |0\rangle :: \texttt{Repeat}\,n \\ \multicolumn{2}{c}{\texttt{BQWalk}\,|1\rangle\,S(n) \to |1\rangle :: \texttt{BQWalk}\,(\texttt{Had}\,|1\rangle)\,n} \end{array}\right\}$$

Coming back on our example

$\texttt{Translate}(\texttt{bqwalk})$ yields the following system:

$$\left\{\begin{array}{rr} \texttt{Repeat}\,0 \to [\,] & \texttt{Repeat}\,S(n) \to |0\rangle :: \texttt{Repeat}\,n \\ \texttt{BQWalk}\,|1\rangle\,0 \to |1\rangle :: [\,] & \texttt{BQWalk}\,|0\rangle\,n \to |0\rangle :: \texttt{Repeat}\,n \\ \multicolumn{2}{c}{\texttt{BQWalk}\,|1\rangle\,S(n) \to |1\rangle :: \texttt{BQWalk}\,(\texttt{Had}\,|1\rangle)\,n} \end{array}\right\}$$

- $\texttt{BQWalk}$ terminates in size $\mathcal{O}(n)$ for inputs of size $\mathcal{O}(n)$.
- Quantum circuit is of polynomial size
- Use existing techniques to get termination and complexity certificates: polynomial / quasi / sup-interpretations, path orderings, size change principle...

4. Future work

Future work

Contributions

- A quantum language with both control flows and general recursion
- Link between runtime-complexity and quantum circuits size
- Translation to TRS, preserving semantics and runtime-complexity

Future steps

- Adapt existing complexity results from TRS to fit in the quantum context
- Characterize complexity classes (both for space and time)
- Provide an actual compilation algorithm

Thanks for your attention !