量子プログラミング言語の研究

静的解析から量子プログラム統合開発環境へ

国立情報学研究所
総合研究大学院大学
ERATO 蓮尾メタ数理システムデザインプロジェクト
蓮尾 一郎

in Collaboration with
崇城大学
星野 直彦
京都大学 情報学研究科
脇坂 遼 ・ 五十嵐 淳

情報処理学会 連続セミナー2020「人間中心社会を支える情報技術の新潮流」
2020/12/14 「量子コンピュータとソフトウェア」

Outline

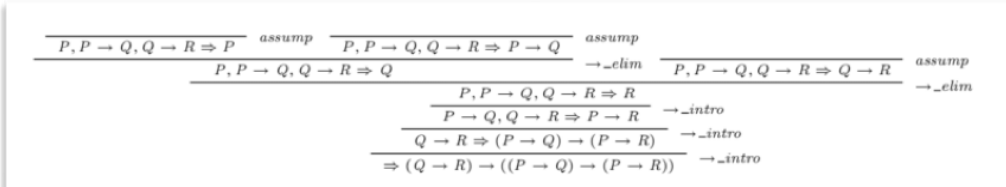
- Introduction
- Hoare logic as a showcase of static analysis
- QPL Work (1): Ying's quantum Hoare logic
- QPL Work (2): Quantum GoI for semantics of quantum λ-calculus
  [H. & Hoshino]
- "Static analysis for hardware-specific, low-level concerns in NISQ"
- QPL Work (3): Quantum Hoare logic for bounding errors [Hung+, POPL'19]
- QPL Work (4): Language-Based Automated Uncomputation [Bichsel+, PLDI'20]
- QPL Work (5): Type System for Qubit Connectivity
  [Wakizaka & Igarashi, JSSST Congress '20]
- Towards Quantum IDE

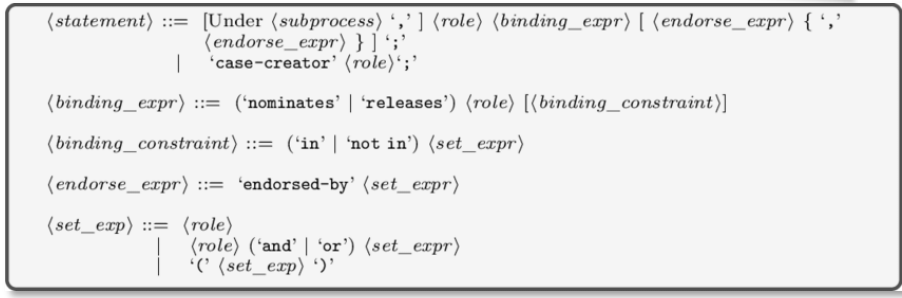
自己紹介

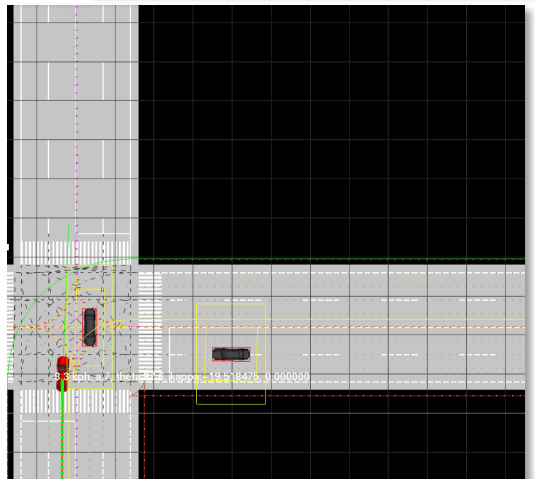
Ichiro Hasuo (NII & SOKENDAI)
PhD (Computer Science, Radboud U. Nijmegen, 2008)

- 研究テーマ： **ソフトウェアの安全性**
  - **形式検証**
    （ソフトウェアが安全であることを定理として証明）
  - **プログラミング言語理論**
    （ソフトウェア記述のための形式体系）
  - **ソフトウェア意味論**
    （ソフトウェアの振舞いの数学的定義．形式検証に必要）
  - **論理学，代数学，圏論**
    （ソフトウェア意味論の数学的道具）
  - **物理情報システム，自動運転，機械学習システム**
    （最近の応用，ERATO プロジェクト）

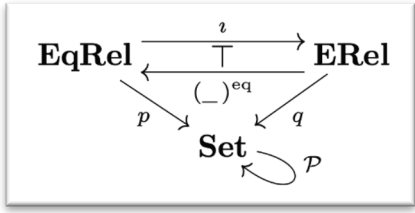
- このトークは，「量子ソフトウェア」の中でも特に
  量子プログラミング言語に注力します







Lopez-Pindato+, CAiSE'19



Disclaimer: “Informed Outsider” です

- 量子ソフトウェアとの関わりは
  しばらく前
  - H. & Naohiko Hoshino,
    Semantics of Higher-Order Quantum Computation via Geometry of Interaction.
    LICS’11 & Ann. Pure Appl. Logic ’17
  - 蓮尾，星野：「量子プログラミング言語」
    情報処理 Vol. 55, No. 7 (2014年6月)
  - Organizing Chair, 11th workshop on Quantum Physics and Logic (QPL 2014), Kyoto, Japan

- 当時の動機：
  数学的一般論のわかりやすい応用例として

特集 量子コンピュータ

5

量子プログラミング言語

蓮尾一郎（東京大学大学院情報理工学系研究科コンピュータ科学専攻）
星野直彦（京都大学数理解析研究所）

量子プログラミング言語とは―研究の動機

量子計算の分野においてアルゴリズムを記述するためには，図 -1 のような量子回路を用いるのが一般的だろう（高橋氏の記事参照．メーターの絵は量子測定を表す）．一方で，（量子でない）古典アルゴリズムを回路で表現することはあまりなく**擬似コード**とよばれるプログラムもどきが用いられる．この一点においてすでに「量子計算のためのプログラミング言語とはどのようなものか?」という疑問が自然に浮かびあがるのであり，筆者らはこの疑問に答えるべく（おもに数学的なアプローチから）研究を行っている．しかし以下ではさらにもう少し，「量子プログラミング言語の実現によって何が可能となるのか?」ということについて論じたい．

論理的記述と物理的実装の分離

まず 1 つの利点として，「高レベルの」アルゴリズムの記述と「低レベルの」物理的実装との分離が挙げられる．（古典計算のための）高級プログラミング言語においては，高レベルのプログラムがコンパイラによって低レベルの機械語コードに変換されたのち実行されるため，プログラマは実行環境のアーキテクチャを意識する必要はない．さらにコンパイラによって数々の最適

図 -1　量子回路の例（量子テレポーテーションと呼ばれる有名なアルゴリズム）

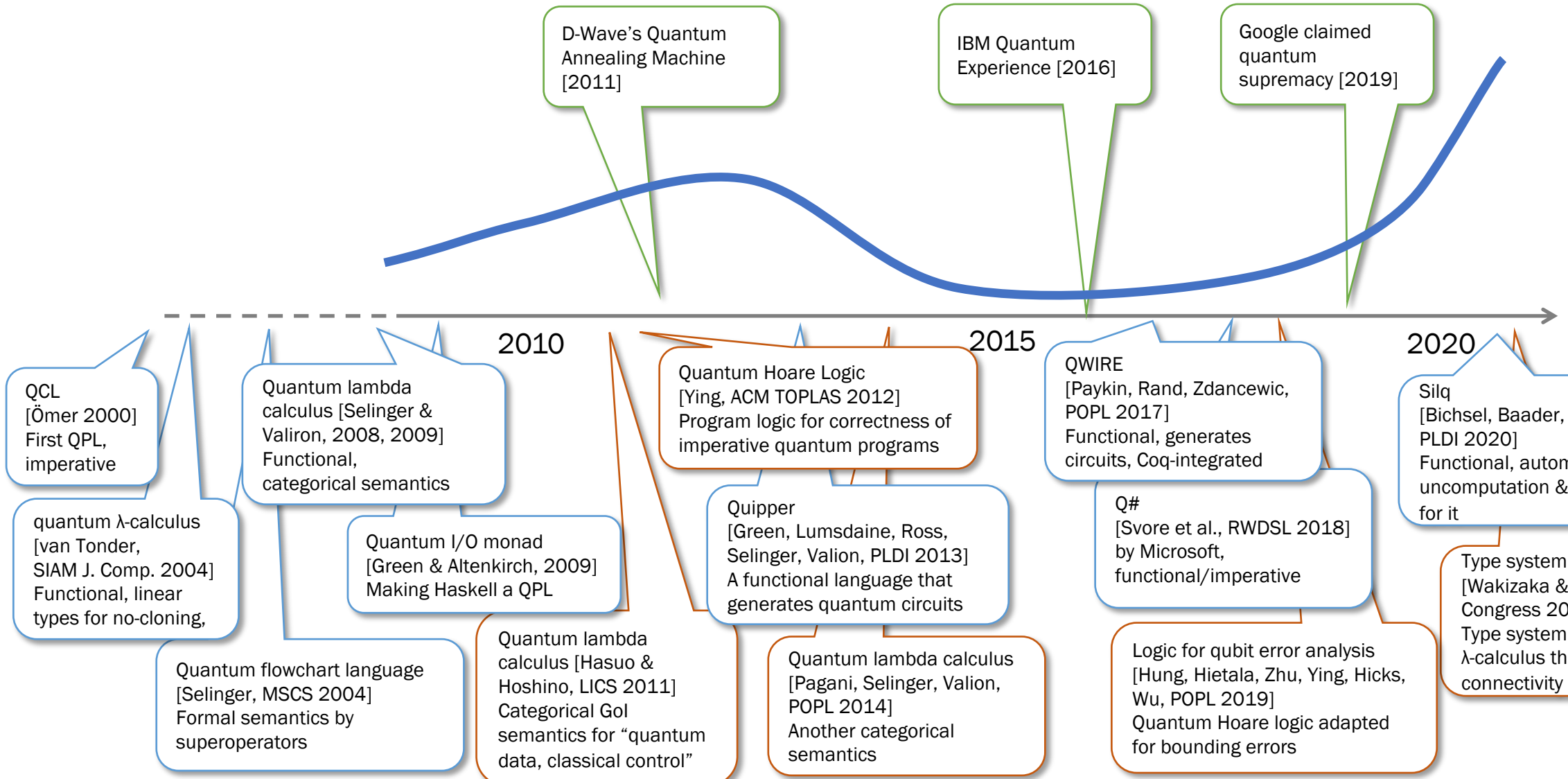
れ）に対処するための膨大な数の誤り訂正をアルゴリズムの記述から隠ぺいできる．
- 量子ビットの再利用などの最適化をコンパイラが自動的に行える．

意味論と仕様検証

次の利点として，量子アルゴリズムに対する**仕様検証**を挙げる．ここで仕様とは「アルゴリズムがみたすべき性質」のことを言い，たとえば「量子テレポーテーションによって Bob が受け取る量子ビットは Alice がはじめに持っていたそれと同じである」というような性質である．アルゴリズムと仕様の 2 つを入力として，「みたす（ことが証明できた）」「みたさない（反例がみつかった）」を出力とする問題を仕様検証と呼ぶのであるが，古典計算に対してはこの仕様検証を行うための手法が盛んに研究され，理論計算機科学において**検証**（verification）と呼ばれる一大分野をなしている．

Trends in Quantum Programming Language (QPL) Research

Not at all comprehensive!



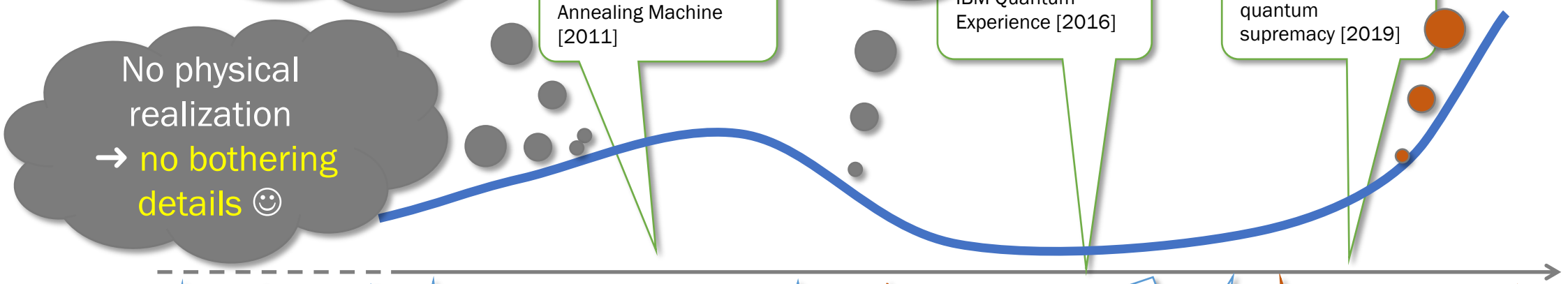
Tr… (QPL)

Chance to exercise advanced PL & semantical techniques…

Theoretically, there is nothing fundamentally challenging…

Physical realization emerging → let's take "details" seriously

No physical realization → no bothering details ☺



QCL [Ömer 2000] First QPL, imperative

quantum λ-calculus [van Tonder, SIAM J. Comp. 2004] Functional, linear types for no-cloning,

Quantum lambda calculus [Selinger & Valiron, 2008, 2009] Functional, categorical semantics

Quantum I/O monad [Green & Altenkirch, 2009] Making Haskell a QPL

Quantum flowchart language [Selinger, MSCS 2004] Formal semantics by superoperators

Quantum lambda calculus [Hasuo & Hoshino, LICS 2011] Categorical GoI semantics for "quantum data, classical control"

Quantum Hoare Logic [Ying, ACM TOPLAS 2012] Program logic for correctness of imperative quantum programs

Quipper [Green, Lumsdaine, Ross, Selinger, Valion, PLDI 2013] A functional language that generates quantum circuits

Quantum lambda calculus [Pagani, Selinger, Valion, POPL 2014] Another categorical semantics

QWIRE [Paykin, Rand, Zdancewic, POPL 2017] Functional, generates circuits, Coq-integrated

Q# [Svore et al., RWDSL 2018] by Microsoft, functional/imperative

Logic for qubit error analysis [Hung, Hietala, Zhu, Ying, Hicks, Wu, POPL 2019] Quantum Hoare logic adapted for bounding errors

Silq [Bichsel, Baader, Gehr, Vechev, PLDI 2020] Functional, automatic uncomputation & a type system for it

Type system for qubit connectivity [Wakizaka & Igarashi, JSSST Congress 2020] Type system for Selinger's quantum λ-calculus that ensures qubit connectivity

Outline

- Introduction
- Hoare logic as a showcase of static analysis
- QPL Work (1): Ying's quantum Hoare logic
- QPL Work (2): Quantum GoI for semantics of quantum λ-calculus
  [H. & Hoshino]
- "Static analysis for hardware-specific, low-level concerns in NISQ"
- QPL Work (3): Quantum Hoare logic for bounding errors [Hung+, POPL'19]
- QPL Work (4): Language-Based Automated Uncomputation [Bichsel+, PLDI'20]
- QPL Work (5): Type System for Qubit Connectivity
  [Wakizaka & Igarashi, JSSST Congress '20]
- Towards Quantum IDE

Hoare Logic

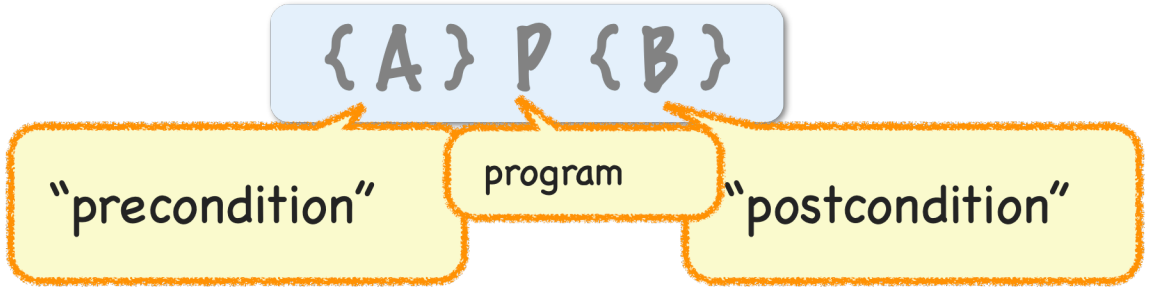
静的解析 static analysis の代表例として

```
n := N;
k := 1;
while (n > 0) {
  k := k*n;
  n := n-1;
}
```

Computes the factorial N! ...

Really? Can we prove it?

- A logical system for program correctness
- Derives Hoare triples

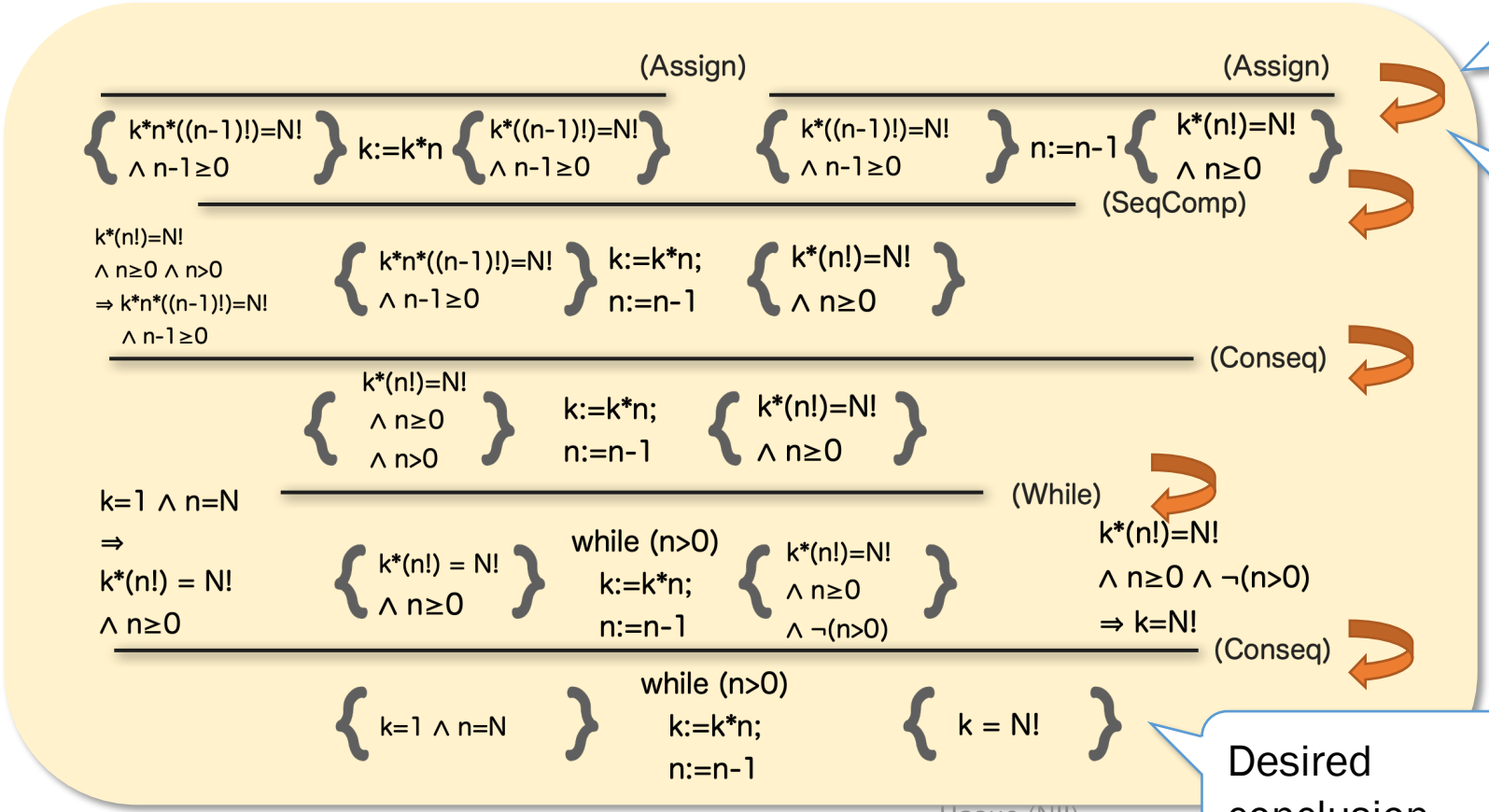


  - Such as

$$\{\ k=1 \wedge n=N\ \}\quad \text{while } (n>0)\ k:=k*n;\ n:=n-1 \quad \{\ k = N!\ \}$$

Hoare Logic

- A derivation ("proof") is repeated application of syntactic derivation rules ➔ automated proof check & proof search



From top to bottom…

Validity of rule application is checked (mostly) syntactically, symbolically, mechanically

Desired conclusion

Outline

- Introduction
- Hoare logic as a showcase of static analysis
  - Program correctness, semantically
  - Syntactic derivation rules (proof system) in Hoare logic
  - Soundness (meta-)theorem, summary
- QPL Work (1): Ying's quantum Hoare logic
- …

Program Correctness, Semantically

```
n := N;
k := 1;
while (n > 0) {
   k := k*n;
   n := n-1;
}
```

Computes the factorial N!

Claim: After the execution, k= N!

Difficulty:
No idea how many iterations

Idea: Properties preserved before/after the loop (loop invariant)

Program Correctness, Semantically

```
n := N;
k := 1;
while (n > 0) {
   k := k*n;
   n := n-1;
}
```

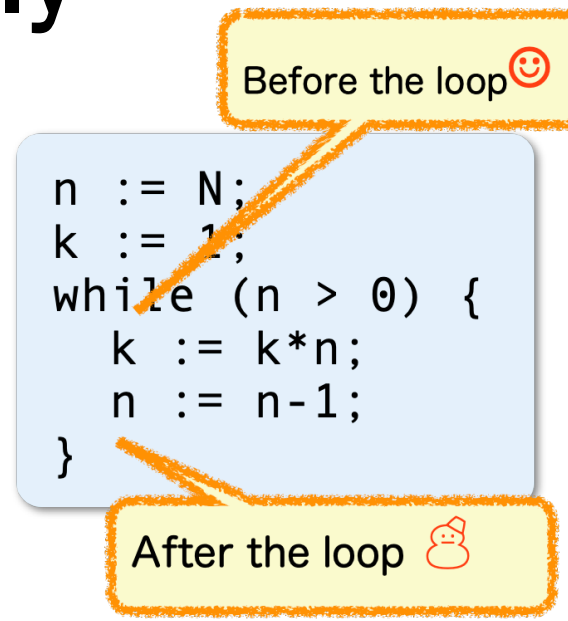
Idea: Properties preserved before/after the loop

(loop invariant)

$k * (n!) = N!$

	k	n
✔	1	N
✔	N	N-1
✔	N*(N-1)	N-2
✔	N*(N-1)*(N-2)	N-3
✔	...	...
✔	N*(N-1)*...*2	1
✔	N*(N-1)*...*2*1	0

Program Correctness, Semantically



Loop invariant:

$k * (n!) = N!$

Lem. 1

$k * (n!) = N!$ is indeed a loop invariant.

That is: if true at ☺, then true at ☃.

Proof:

k_{old}, n_{old} and k_{new}, n_{new} : values of k, n before/after the loop Then:

(Asmp. ☺) $k_{old} * (n_{old}!) = N!$

(Aim) $k_{new} * (n_{new}!) = N!$

Now:

$k_{new} * (n_{new}!)$

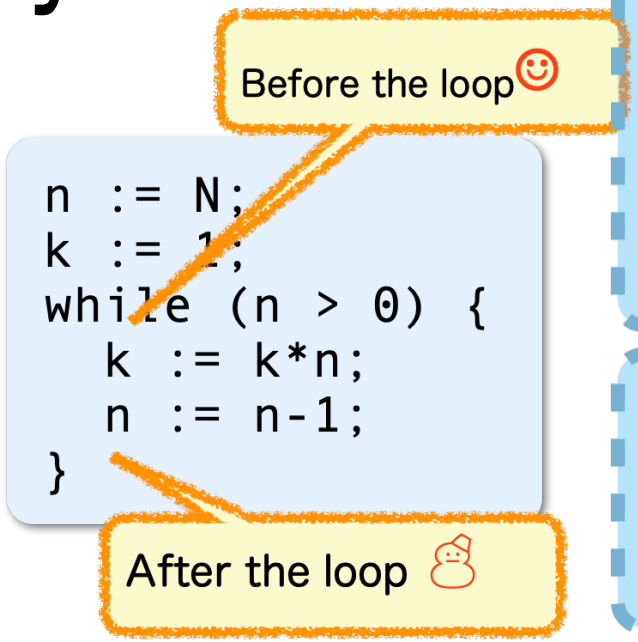
$= (k_{old} * n_{old}) * ((n_{old} - 1)!)$ [By def. of k_{new}, n_{new}]

$= k_{old} * (n_{old}!)$

$= N!$ [By asmp. ☺]

□

Program Correctness, Semantically



Lem. 1

k * (n!) = N! is indeed a loop invariant.

That is: if true at ☺, then true at .

Lem. 2

k * (n!) = N! is true, before the loop, at ☺.

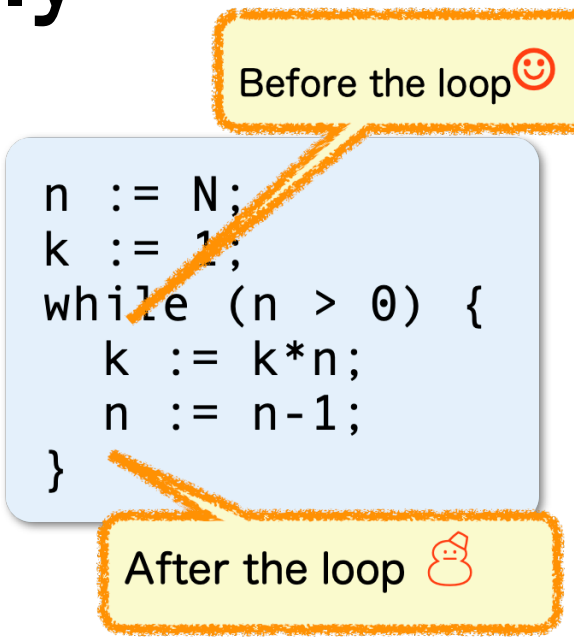
Loop invariant:

$$k * (n!) = N!$$

Proof:

Obvious from n = N, k = 1. □

Program Correctness, Semantically



Lem. 1

k * (n!) = N! is indeed a loop invariant.

That is: if true at ☺, then true at ☃.

Lem. 2

k * (n!) = N! is true, before the loop, at ☺.

Loop invariant:

$$k * (n!) = N!$$

Proof of correctness:

By Lem. 1 & 2, k * (n!) = N! holds before/during/after the loop.

When out of the loop we have n=0; this forces k = N!. □

Outline

- Introduction
- Hoare logic as a showcase of static analysis
  - Program correctness, semantically
  - Syntactic derivation rules (proof system) in Hoare logic
  - Soundness (meta-)theorem, summary
- QPL Work (1): Ying's quantum Hoare logic
- …

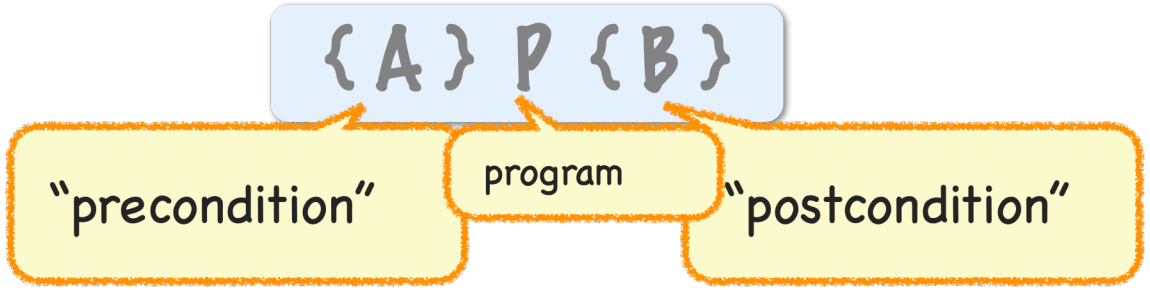
Hoare Logic

```
n := N;
k := 1;
while (n > 0) {
  k := k*n;
  n := n-1;
}
```

Computes the factorial N! …

Really? Can we prove it?

- A logical system for program correctness
- Derives Hoare triples



  - Such as

$$\{\ k=1 \wedge n=N\ \}\quad \text{while } (n>0)\ k:=k*n;\ n:=n-1 \quad \{\ k = N!\ \}$$

Hoare Logic: (Syntactic) Derivation Rules

$$\frac{}{\{ A[a/x] \}\ x:=a\ \{ A \}}\ \text{(Assign)}$$

$$\frac{\{ A \}\ P_1\ \{ C \} \qquad \{ C \}\ P_2\ \{ B \}}{\{ A \}\ P_1; P_2\ \{ B \}}\ \text{(SeqComp)}$$

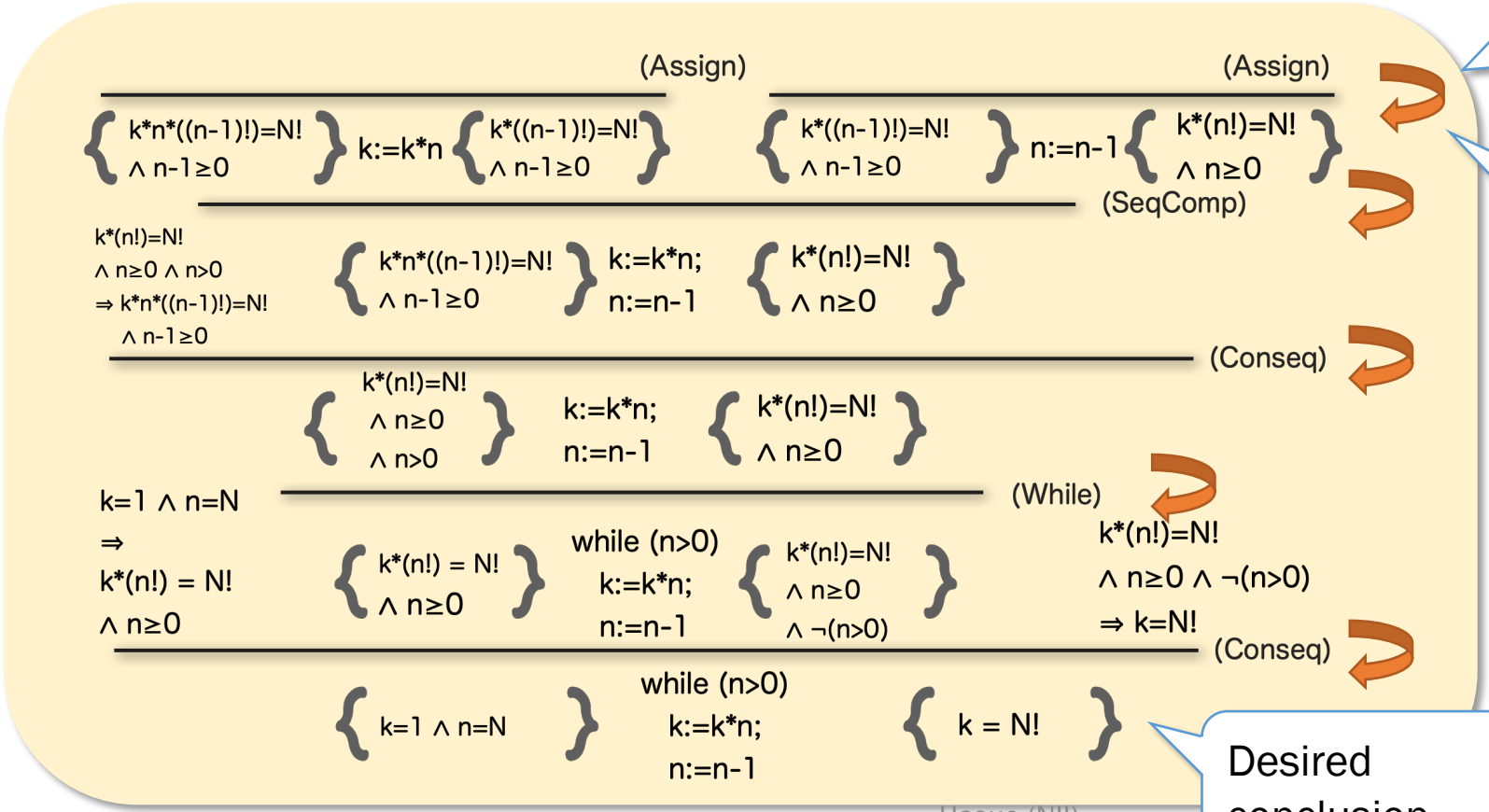
$$\frac{\{ A \wedge b \}\ P_1\ \{ B \} \qquad \{ A \wedge \neg b \}\ P_2\ \{ B \}}{\{ A \}\ \text{if } b \text{ then } P_1 \text{ else } P_2\ \{ B \}}\ \text{(If)}$$

$$\frac{\{ A \wedge b \}\ P_1\ \{ A \}}{\{ A \}\ \text{while } b\ P_1\ \{ A \wedge \neg b \}}\ \text{(While)}$$

$$\frac{A \Rightarrow A' \qquad \{ A' \}\ P\ \{ B' \} \qquad B' \Rightarrow B}{\{ A \}\ P\ \{ B \}}\ \text{(Conseq)}$$

Hoare Logic

- A derivation ("proof") is repeated application of syntactic derivation rules ➔ automated proof check & proof search



From top to bottom…

Validity of rule application is checked (mostly) syntactically, symbolically, mechanically

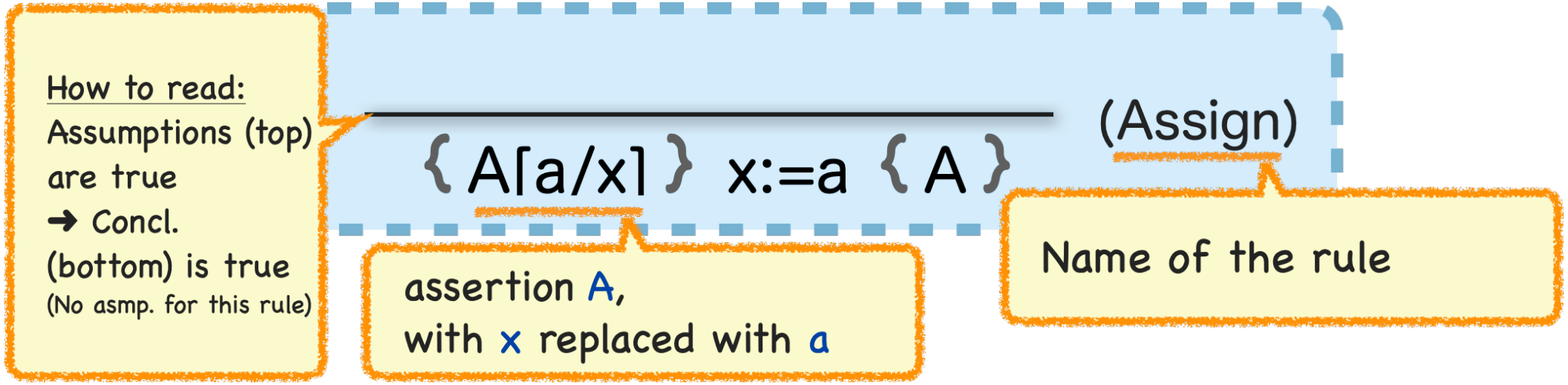
Desired conclusion

Outline

- Introduction
- Hoare logic as a showcase of static analysis
  - Program correctness, semantically
  - Syntactic derivation rules (proof system) in Hoare logic
  - Soundness (meta-)theorem, summary
- QPL Work (1): Ying's quantum Hoare logic
- …

Are These Rules "Correct"?

- For example:



  - "To have A after x:=a, we need to have A[a/x] before"
  - Examples:

$$\frac{}{\{ y=2 \}\ x:=y\ \{ x=2 \}} \qquad \frac{}{\{ x-1=2 \}\ x:=x-1\ \{ x=2 \}}$$

Semantics (Meaning) and Correctness

- Def. A *memory state* is a function $\sigma : \mathbf{Var} \longrightarrow \mathbb{Z}$
- Def. *Program semantics* $[\![P]\!] : \mathbf{MSt} \longrightarrow \mathbf{MSt} \cup \{\bot\}$ is a transformation of memory states, defined inductively on the construction of a program P

$$[\![\mathtt{x} := a]\!] : \quad \sigma \quad \longmapsto \quad \sigma\big[\mathtt{x} \mapsto [\![a]\!]\sigma\big]$$

$$[\![P_1 ; P_2]\!] : \quad \sigma \quad \longmapsto \quad [\![P_2]\!]([\![P_1]\!]\sigma)$$

$$[\![\mathtt{if}\ b\ \mathtt{then}\ P_1\ \mathtt{else}\ P_2]\!] :$$

$$\sigma \quad \longmapsto \quad \begin{cases} [\![P_1]\!]\sigma & \text{if } [\![b]\!]\sigma \text{ is true} \\ [\![P_2]\!]\sigma & \text{if } [\![b]\!]\sigma \text{ is false} \end{cases}$$

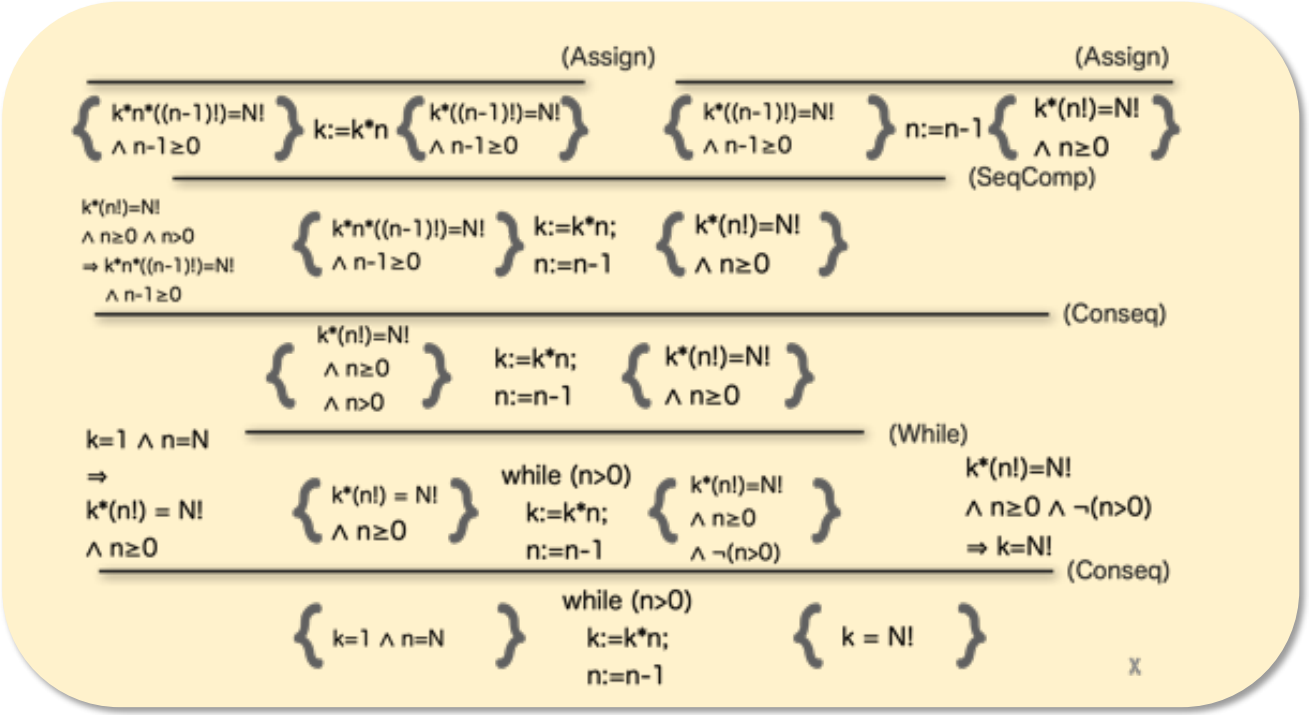
$$[\![\mathtt{while}\ b\ P]\!] : \quad \sigma \quad \longmapsto \quad \ldots$$

- Def. A Hoare triple {A}P{B} is *true* if, for any memory state σ that satisfies A, $[\![P]\!]\sigma$ satisfies B

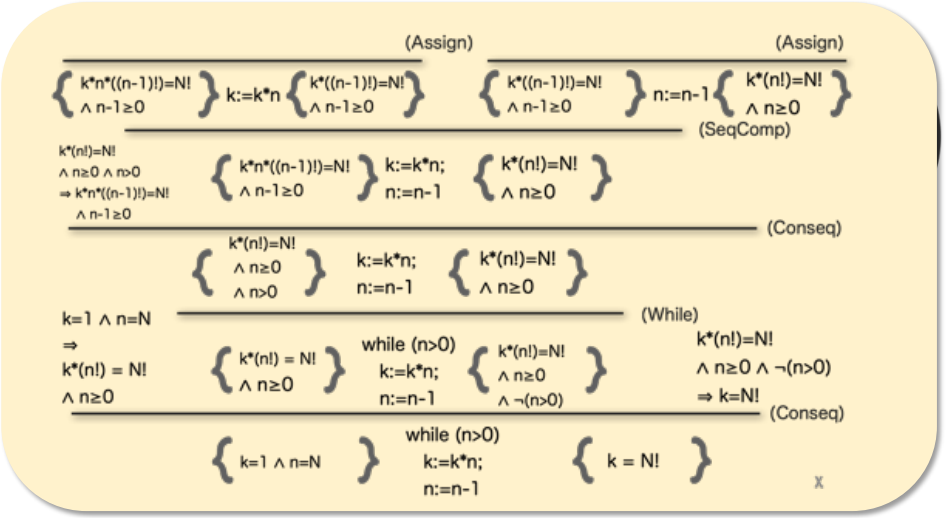
Soundness (Meta-)Theorem

- **Thm.** In each rule application of Hoare logic, if the premises are true, then the conclusion is true

- **Cor.** (Soundness)
  Every Hoare triple that is derived in Hoare logic is true



Hoare Logic: Summary



- As an example of static analysis techniques…
- Points
  - Static (⇔ dynamic): reasoning about programs without executing them
  - Reasoning about infinitely many input values all at once (N can be any integer)
  - Syntactic, mechanizable. Potential automation
  - The goal can be other than correctness (as we'll see)
  - Symbol manipulation
    ➜ need meta-theory (soundness etc.) to make sense of it

Outline

- Introduction
- Hoare logic as a showcase of static analysis
- QPL Work (1): Ying's quantum Hoare logic
- QPL Work (2): Quantum GoI for semantics of quantum λ-calculus [H. & Hoshino]
- "Static analysis for hardware-specific, low-level concerns in NISQ"
- QPL Work (3): Quantum Hoare logic for bounding errors [Hung+, POPL'19]
- QPL Work (4): Language-Based Automated Uncomputation [Bichsel+, PLDI'20]
- QPL Work (5): Type System for Qubit Connectivity [Wakizaka & Igarashi, JSSST Congress '20]
- Towards Quantum IDE

Quantum Hoare Logic [Ying, ACM TOPLAS 2012]

- Natural extension of Hoare logic
- Target language: imperative
  - Variables ➜ quantum variables
  - Assignment
    ➜ application of unitary gate
  - If-branch ➜ branching by measurement
- Assertions: use projections
  - Classical: x = y + 1, ∃y. (x = 2y), x > y, …
  - Quantum: a self-adjoint operator P such that $\mathbf{0} \sqsubseteq \boldsymbol{P} \sqsubseteq \boldsymbol{I}$
- Semantics: (as expected)

Target programming language

$$S ::= \textbf{skip} \mid q := 0 \mid \overline{q} := U\overline{q} \mid S_1; S_2 \mid \textbf{measure}\, M[\overline{q}] : \overline{S} \mid \textbf{while}\, M[\overline{q}] = 1\, \textbf{do}\, S$$

Quantum Hoare Logic [Ying, ACM TOPLAS 2012]

- Derivation rules mirror classical ones

$(Axiom\ Skip) \qquad \{P\}\mathbf{Skip}\{P\}$

$(Axiom\ Initialization)$ (1) If $type(q) = \mathbf{Boolean}$, then

$$\{|0\rangle_q\langle 0|P|0\rangle_q\langle 0| + |1\rangle_q\langle 0|P|0\rangle_q\langle 1|\}q := 0\{P\}$$

(2) If $type(q) = \mathbf{integer}$, then

$$\{\sum_{n=-\infty}^{\infty} |n\rangle_q\langle 0|P|0\rangle_q\langle n|\}q := 0\{P\}$$

$(Axiom\ Unitary\ Transformation) \qquad \{U^\dagger PU\}\overline{q} := U\overline{q}\{P\}$

$$(Rule\ Sequential\ Composition) \qquad \frac{\{P\}S_1\{Q\} \quad \{Q\}S_2\{R\}}{\{P\}S_1; S_2\{R\}}$$

$$(Rule\ Measurement) \qquad \frac{\{P_m\}S_m\{Q\} \text{ for all } m}{\{\sum_m M_m^\dagger P_m M_m\}\mathbf{measure}\ M[\overline{q}] : \overline{S}\{Q\}}$$

$$(Rule\ Loop\ Partial) \qquad \frac{\{Q\}S\{M_0^\dagger PM_0 + M_1^\dagger QM_1\}}{\{M_0^\dagger PM_0 + M_1^\dagger QM_1\}\mathbf{while}\ M[\overline{q}] = 1\ \mathbf{do}\ S\{P\}}$$

$$(Rule\ Order) \qquad \frac{P \sqsubseteq P' \quad \{P'\}S\{Q'\} \quad Q' \sqsubseteq Q}{\{P\}S\{Q\}}$$

- Case study: correctness of the Grover search algorithm
  “after execution, a solution is found with at least this much probability”

Outline

- Introduction
- Hoare logic as a showcase of static analysis
- QPL Work (1): Ying's quantum Hoare logic
- QPL Work (2): Quantum GoI for semantics of quantum λ-calculus [H. & Hoshino]
- "Static analysis for hardware-specific, low-level concerns in NISQ"
- QPL Work (3): Quantum Hoare logic for bounding errors [Hung+, POPL'19]
- QPL Work (4): Language-Based Automated Uncomputation [Bichsel+, PLDI'20]
- QPL Work (5): Type System for Qubit Connectivity [Wakizaka & Igarashi, JSSST Congress '20]
- Towards Quantum IDE

Quantum Geometry of Interaction

[H. & Hoshino, LICS'11 & Ann. Pure Appl. Logic '17]

- Semantics of higher-order functional QPL

Quantum teleportation: $\lambda x^{\mathtt{qbit}}.\ \mathtt{corr}\left(\mathtt{Bellmeasure}\left(\ \mathtt{cmp}_{1,2}\ \langle x, \mathtt{EPR}\rangle\ \right)\right)$

where

$$\mathtt{EPR} :\equiv U_0\left(\mathtt{cmp}_{1,1}\langle \mathtt{new}_{|0\rangle\langle 0|}, \mathtt{new}_{|0\rangle\langle 0|}\rangle\right) \quad : \quad 2\text{-}\mathtt{qbit}$$

$$\mathtt{Bellmeasure} :\equiv \lambda w^{3\text{-}\mathtt{qbit}}.\ \mathtt{let}\ \langle b_0^{\mathtt{bit}}, p^{2\text{-}\mathtt{qbit}}\rangle = \mathtt{meas}_1^3\,(U_1 w)\ \mathtt{in}$$
$$\mathtt{let}\ \langle b_1^{\mathtt{bit}}, q^{\mathtt{qbit}}\rangle = \mathtt{meas}_1^2\, p\ \mathtt{in}\ \langle b_0, \langle b_1, q\rangle\rangle$$
$$: \quad 3\text{-}\mathtt{qbit} \multimap \mathtt{bit} \boxtimes (\mathtt{bit} \boxtimes \mathtt{qbit})$$

$$\mathtt{corr} :\equiv \lambda x^{\mathtt{bit}\boxtimes(\mathtt{bit}\boxtimes\mathtt{qbit})}.\ \mathtt{let}\ \langle b_0^{\mathtt{bit}}, y^{\mathtt{bit}\boxtimes\mathtt{qbit}}\rangle = x\ \mathtt{in}\ \mathtt{let}\ \langle b_1^{\mathtt{bit}}, q^{\mathtt{qbit}}\rangle = y\ \mathtt{in}$$
$$\mathtt{match}\ b_0\ \mathtt{with}\ ($$
$$z_0^\top \mapsto \mathtt{match}\ b_1\ \mathtt{with}\ (w_0^\top \mapsto U_{00}q \mid w_1^\top \mapsto U_{01}q)$$
$$\mid z_1^\top \mapsto \mathtt{match}\ b_1\ \mathtt{with}\ (w_0^\top \mapsto U_{10}q \mid w_1^\top \mapsto U_{11}q))$$
$$: \quad \mathtt{bit} \boxtimes (\mathtt{bit} \boxtimes \mathtt{qbit}) \multimap \mathtt{qbit}$$

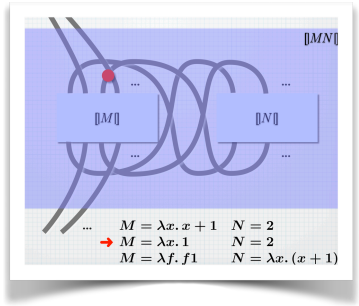
Types from linear logic for enhanced resource analysis (qubits cannot be used twice ➜ no-cloning)

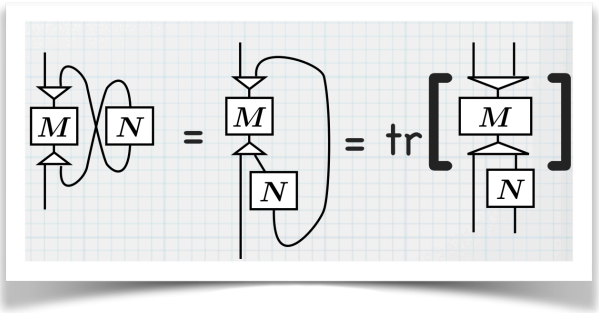
- "Quantum data, classical control"—much like other QPLs
- Semantics of higher-order functional languages is challenging already in classical settings (λ-calculus, domain theory, categorical semantics, …)

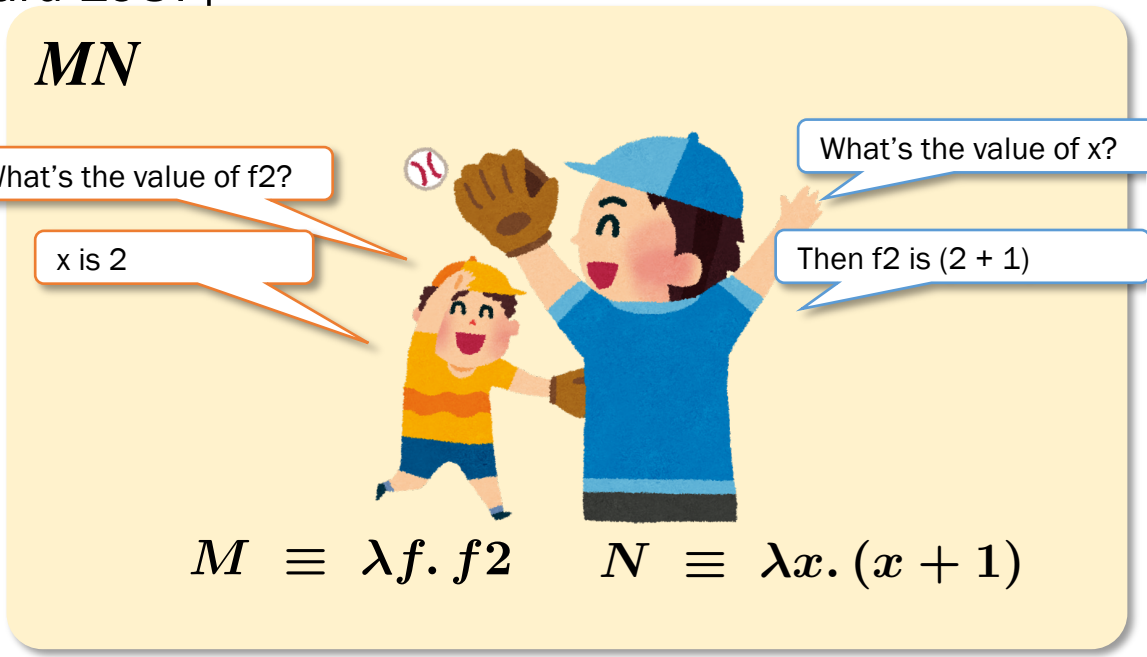
Quantum Geometry of Interaction

[H. & Hoshino, LICS'11 & Ann. Pure Appl. Logic '17]

- Tool: geometry of interaction (GoI) [Girard 1987]
  - Higher-order computation interpreted by first-order interaction
  - Categorical axiomatization by traced monoidal categories
    [Abramsky, Haghverdi, Scott 2002]







- Different base categories for different effects (nondeterminism, probability, quantum). The essence is the same

Outline

- Introduction
- Hoare logic as a showcase of static analysis
- QPL Work (1): Ying's quantum Hoare logic
- QPL Work (2): Quantum GoI for semantics of quantum λ-calculus
  [H. & Hoshino]
- "Static analysis for hardware-specific, low-level concerns in NISQ"
- QPL Work (3): Quantum Hoare logic for bounding errors [Hung+, POPL'19]
- QPL Work (4): Language-Based Automated Uncomputation [Bichsel+, PLDI'20]
- QPL Work (5): Type System for Qubit Connectivity
  [Wakizaka & Igarashi, JSSST Congress '20]
- Towards Quantum IDE

Static Analysis: Towards Low-Level Concerns

- Static analysis: reasoning about programs without execution
  - Goals: correctness (in Hoare logic, …),
    security (non-interference [Sabelfeld & Myers '03], differential privacy [Barthe+ '14], …),
    complexity analysis [Hofmann'00], …
- For quantum programs?
  - Execution is costly ➔ static analysis makes even bigger sense
  - Can be seen as generalization of classical simulation
  - Hardware constraints (such as in NISQ)
    ➔ a number of new goals of static analysis
    (error estimate, uncomputation, qubit connectivity, …)
- "Static analysis for hardware-specific, low-level concerns in NISQ"

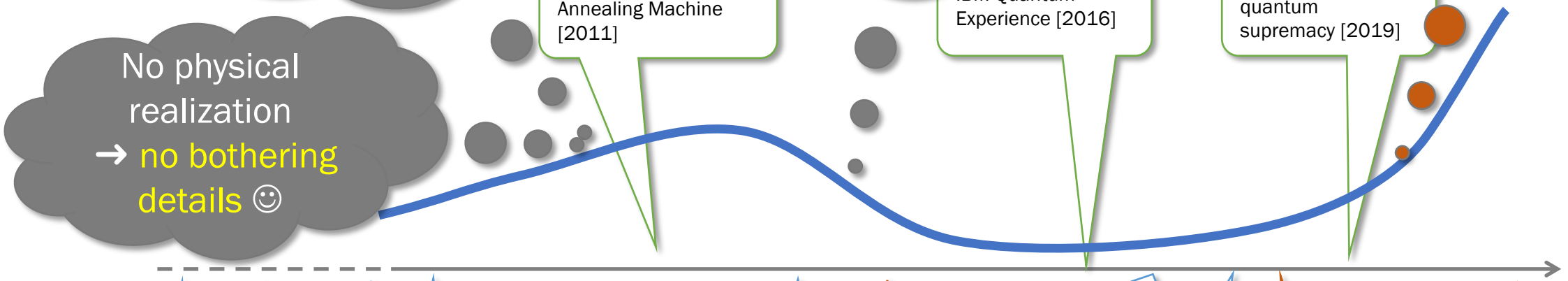
Tr… (QPL)

Chance to exercise advanced PL & semantical techniques…

Theoretically, there is nothing fundamentally challenging…

Physical realization emerging → let's take "details" seriously

No physical realization → no bothering details ☺



D-Wave's Quantum Annealing Machine [2011]

IBM Quantum Experience [2016]

Google claimed quantum supremacy [2019]

QCL [Ömer 2000] First QPL, imperative

quantum λ-calculus [van Tonder, SIAM J. Comp. 2004] Functional, linear types for no-cloning,

Quantum lambda calculus [Selinger & Valiron, 2008, 2009] Functional, categorical semantics

Quantum I/O monad [Green & Altenkirch, 2009] Making Haskell a QPL

Quantum flowchart language [Selinger, MSCS 2004] Formal semantics by superoperators

Quantum Hoare Logic [Ying, ACM TOPLAS 2012] Program logic for correctness of imperative quantum programs

Quipper [Green, Lumsdaine, Ross, Selinger, Valion, PLDI 2013] A functional language that generates quantum circuits

Quantum lambda calculus [Hasuo & Hoshino, LICS 2011] Categorical GoI semantics for "quantum data, classical control"

Quantum lambda calculus [Pagani, Selinger, Valion, POPL 2014] Another categorical semantics

QWIRE [Paykin, Rand, Zdancewic, POPL 2017] Functional, generates circuits, Coq-integrated

Q# [Svore et al., RWDSL 2018] by Microsoft, functional/imperative

Logic for qubit error analysis [Hung, Hietala, Zhu, Ying, Hicks, Wu, POPL 2019] Quantum Hoare logic adapted for bounding errors

Silq [Bichsel, Baader, Gehr, Vechev, PLDI 2020] Functional, automatic uncomputation & a type system for it

Type system for qubit connectivity [Wakizaka & Igarashi, JSSST Congress 2020] Type system for Selinger's quantum λ-calculus that ensures qubit connectivity

Outline

- Introduction
- Hoare logic as a showcase of static analysis
- QPL Work (1): Ying's quantum Hoare logic
- QPL Work (2): Quantum GoI for semantics of quantum λ-calculus
  [H. & Hoshino]
- "Static analysis for hardware-specific, low-level concerns in NISQ"
- QPL Work (3): Quantum Hoare logic for bounding errors [Hung+, POPL'19]
- QPL Work (4): Language-Based Automated Uncomputation [Bichsel+, PLDI'20]
- QPL Work (5): Type System for Qubit Connectivity
  [Wakizaka & Igarashi, JSSST Congress '20]
- Towards Quantum IDE

Quantum Hoare Logic for Bounding Errors

[Hung, Hietala, Zhu, Ying, Hicks, Wu, POPL'19]

- Target language:
  Ying's imperative quantum QPL + noisy unitary gates

Applies U with prob. 1-p; applies Φ with prob. p (error)

$$[\![\overline{q} :\cong_{p,\Phi} U[\overline{q}]]\!]\rho = (1-p)U\rho U^\dagger + p\,\Phi(\rho)$$

$$\widetilde{P} ::= \textbf{skip} \mid q := |0\rangle \mid \overline{q} :\cong_{p,\Phi} U[\overline{q}] \mid \widetilde{P_1}; \widetilde{P_2} \mid$$

$$\textbf{case } M[\overline{q}] = \overline{m \to \widetilde{P_m}} \textbf{ end} \mid \textbf{while } M[\overline{q}] = 1 \textbf{ do } \widetilde{P_1} \textbf{ done}$$

Quantum Hoare Logic for Bounding Errors

[Hung, Hietala, Zhu, Ying, Hicks, Wu, POPL'19]

- The logic extends Ying's quantum Hoare logic with "error bounding judgments"

$$(Q, \lambda) \vdash \widetilde{P} \leq \epsilon$$

If the current quantum state ρ satisfies tr(Qρ) ≧ λ,
then the trace distance between
- the result of the noisy program P and
- that of the ideal program P

is at most ε (the actual def. involves ancillas)

- Derivation rules are introduced, such as

$$\frac{(Q_1, \lambda) \vdash \widetilde{P_1} \leq \epsilon_1 \qquad (Q_2, \lambda) \vdash \widetilde{P_2} \leq \epsilon_2 \qquad \{Q_1\} P_1 \{Q_2\}}{(Q_1, \lambda) \vdash (\widetilde{P_1}; \widetilde{P_2}) \leq \epsilon_1 + \epsilon_2} \text{ (Sequence)}$$

Quantum Hoare Logic for Bounding Errors

[Hung, Hietala, Zhu, Ying, Hicks, Wu, POPL'19]

- They also show that, for the base case (noisy unitary gate), error bounds can be computed by solving suitable SDP

Let $\mathcal{E}$ and $\mathcal{E}'$ be superoperators over $\mathcal{H}$. By extending Watrous [2009], we show that $\|\mathcal{E}-\mathcal{E}'\|_{Q,\lambda}$ can be efficiently computed by the following semidefinite program (SDP):

$$\begin{aligned} \max \quad & \mathrm{tr}(J(\Phi)W) && (6.5)\\ \text{s.t.} \quad & W \le I_{\mathcal{H}} \otimes \rho,\ \mathrm{tr}(Q\rho) \ge \lambda, && (6.6)\\ & \rho \in \mathcal{D}(\mathcal{H}),\ W \text{ is a positive semidefinite operator over } \mathcal{H}\otimes\mathcal{H}, && (6.7) \end{aligned}$$

where $\Phi = \mathcal{E} - \mathcal{E}'$ and $J(\Phi)$ is the *Choi-Jamiolkowski* representation of Φ.

$$\frac{\|U \circ U^\dagger - \Phi\|_{Q,\lambda} \le \epsilon}{(Q,\lambda) \vdash (\overline{q} :\cong_{p,\Phi} U[\overline{q}]) \le p\epsilon}\ (\text{Unitary})$$

Quantum Hoare Logic for Bounding Errors

[Hung, Hietala, Zhu, Ying, Hicks, Wu, POPL'19]

- Examples:
  quantum Bernoulli factory (QBF)

$$\begin{aligned}\widetilde{QBF} \equiv\ & q_1 := |1\rangle;\ q_2 := |1\rangle;\\ & \textbf{while } M[q_2] = 1 \textbf{ do}\\ & \quad q_1 := |0\rangle;\ q_2 := |0\rangle;\\ & \quad q_1 :\cong_{p_V,\Phi_V} V[q_1];\ q_2 :\cong_{p_V,\Phi_V} V[q_2];\\ & \quad q_1, q_2 :\cong_{p_U,\Phi_U} U[q_1, q_2] \textbf{ done},\end{aligned}$$

Classical computation
(symbolic reasoning + SDP)

$$(I, 0) \vdash \widetilde{QBF} \leq 4\epsilon_V + 2\epsilon_U$$

O(p)
if $p = p_V = p_U$

"For ρ s.t. tr(Iρ) ≧ 0"
→ no constraint

where

$$\epsilon_V = p_V \|\Phi_V - V \circ V^\dagger\|_\diamond \text{ and } \epsilon_U = p_U \|\Phi_U - U \circ U^\dagger\|_\diamond$$

Fault-tolerant QBF

ENCODE, CORRECT:
3-qubit repetition code

$$\begin{aligned}\widetilde{\overline{QBF}} \equiv & q_1 := |1\rangle;\ q_2 := |1\rangle;\\ & \textbf{while } M[q_2] = 1 \textbf{ do}\\ & \quad q_1 := |0\rangle;\ q_2 := |0\rangle;\\ & \quad \overline{q}_1 := ENCODE[q_1];\ \overline{q}_2 := ENCODE[q_2];\\ & \quad \overline{q}_1 :\cong_{1,\Phi_{\overline{V}}} \overline{V}[\overline{q}_1];\\ & \quad \overline{q}_1 := CORRECT[\overline{q}_1];\\ & \quad \overline{q}_2 :\cong_{1,\Phi_{\overline{V}}} \overline{V}[\overline{q}_2];\\ & \quad \overline{q}_2 := CORRECT[\overline{q}_2];\\ & \quad \overline{q}_1, \overline{q}_2 :\cong_{1,\Phi_{\overline{U}}} \overline{U}[\overline{q}_1, \overline{q}_2];\\ & \quad \overline{q}_1 := CORRECT[\overline{q}_1];\ \overline{q}_2 := CORRECT[\overline{q}_2];\\ & \quad q_1 := DECODE[\overline{q}_1];\ q_2 := DECODE[\overline{q}_2]\\ & \textbf{done}\end{aligned}$$

Classical computation
(symbolic reasoning + SDP)

$$(I, 0) \vdash \widetilde{\overline{QBF}} \leq 4\epsilon_{\overline{V}} + 2\epsilon_{\overline{U}}$$

$O(p^2)$
if $p = p_V = p_U$

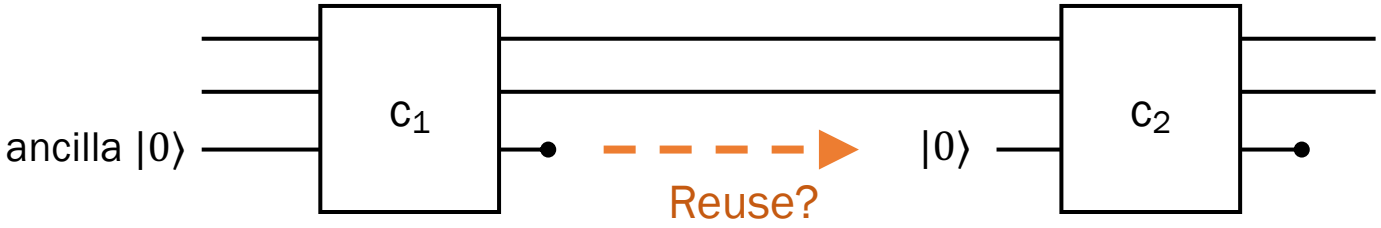
Outline

- Introduction
- Hoare logic as a showcase of static analysis
- QPL Work (1): Ying's quantum Hoare logic
- QPL Work (2): Quantum GoI for semantics of quantum λ-calculus [H. & Hoshino]
- "Static analysis for hardware-specific, low-level concerns in NISQ"
- QPL Work (3): Quantum Hoare logic for bounding errors [Hung+, POPL'19]
- QPL Work (4): Language-Based Automated Uncomputation [Bichsel+, PLDI'20]
- QPL Work (5): Type System for Qubit Connectivity [Wakizaka & Igarashi, JSSST Congress '20]
- Towards Quantum IDE

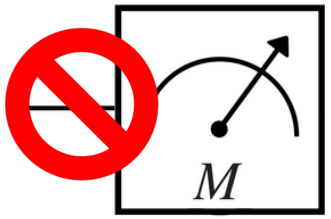
Language-Based Automated Uncomputation

[Bichsel, Baader, Gehr, Vechev, PLDI'20]

- Physical qubits are scarce → ancilla qubits should be reused



- But old ancilla shouldn't be projected out (affects other qubits in entanglement)



- → Uncomputation!
  - Make operations on ancilla qubits reversible, and
  - Apply their inverse to recover $|0\rangle$

Language-Based Automated Uncomputation

[Bichsel, Baader, Gehr, Vechev, PLDI'20]

- Goal: free (quantum) programmers from tedious task of writing uncomputation code…
- by automatic uncomputation
  - Detect reversible operations on ancilla qubits, and
  - Insert uncomputation code

```
1  cTri := 0:int[rrbar];
2  for j in [0..rrbar) {
3    for k in [j+1..rrbar) {
4      if ee[tau[j]][tau[k]]
5         && eew[j] && eew[k]{
6        cTri += 1;
7  } } }
```

Silq

Code in Silq [Bichsel+ '20].
Uncomputation is implicit and automatic

```
1  cTri <- foldM (\cTri j -> do
2    let tau_j = tau ! j
3    eed <- qinit (intMap_replicate rr False)
4    -- computing eed = ee[tau[j]]
5    (taub,ee,eed) <- a11_FetchE tau_j ee eed
6    cTri <- foldM (\cTri k -> do
7      let tau_k = tau ! k
8      eedd_k <- qinit False
9      -- eedd_k=eed[tau[k]]=ee[tau[j]][tau[k]]
10     (tauc, eed, eedd_k) <- qram_fetch qram tau_k eed eedd_k
11     -- using eedd_k as ctrl
12     cTri <- increment cTri `controlled` eedd_k .&&. (eew ! j)
                .&&. (eew ! k)
13     -- uncomputing eedd_k
14     (tauc, eed, eedd_k) <- qram_fetch qram tau_k eed eedd_k
15     qterm False eedd_k
16     return cTri)
17   cTri [j+1..rrbar-1]
18   -- uncomputing eed
19   (taub,ee,eed) <- a11_FetchE tau_j ee eed
20   qterm (intMap_replicate rr False) eed
21   return cTri)
22 cTri [0..rrbar-1]
```

Explicit uncomputation in Quipper

Outline

- Introduction
- Hoare logic as a showcase of static analysis
- QPL Work (1): Ying's quantum Hoare logic
- QPL Work (2): Quantum GoI for semantics of quantum λ-calculus
  [H. & Hoshino]
- "Static analysis for hardware-specific, low-level concerns in NISQ"
- QPL Work (3): Quantum Hoare logic for bounding errors [Hung+, POPL'19]
- QPL Work (4): Language-Based Automated Uncomputation [Bichsel+, PLDI'20]
- QPL Work (5): Type System for Qubit Connectivity
  [Wakizaka & Igarashi, JSSST Congress '20]
- Towards Quantum IDE

Type System for Qubit Connectivity
[Wakizaka & Igarashi, 日本ソフトウェア科学会 2020]

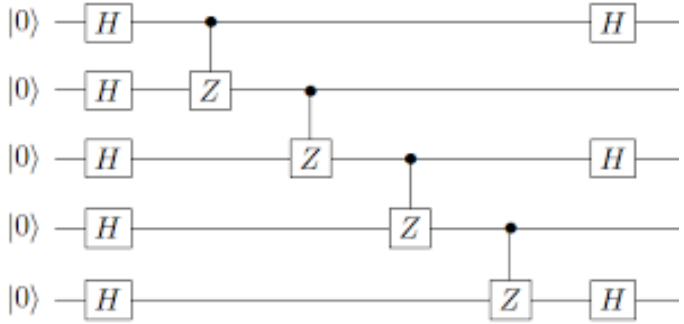
- Approaches to physical qubits

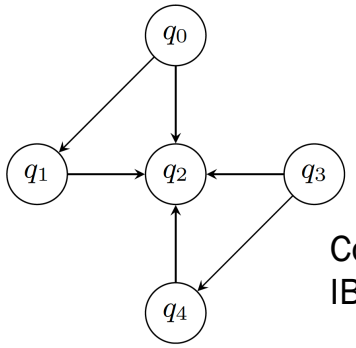
	Scalability	Qubit connectivity	Qubit lifetime	Gate operation time	Players
Superconducting	✓ (c. 50 qubits)	✗ (adjacent only)	✗ (c. 50-100 μs)	✓ (c. 10-50 ns)	Google, IBM, …
Trapped ion	??	✓ (fully connected)	✓ (> 1000 s)	✗ (c. 3-50 μs)	Honeywell, IonQ, …

- Some issues are physical, but others are logical
  → QPL research can help the latter

Type System for Qubit Connectivity
[Wakizaka & Igarashi, 日本ソフトウェア科学会 2020]

- Qubit connectivity:
  binary gates (such as CNOT) can be applied only to connected qubits
- Different hardware has different coupling graphs
- An issue quantum programmers should take care of
- Or better: automated qubit connectivity analysis
  - “Does the given program adhere to qubit connectivity constraints?” (this work)
  - Automatic qubit allocation and swapping





Coupling graph for IBM qx2

Type System for Qubit Connectivity
[Wakizaka & Igarashi, 日本ソフトウェア科学会 2020]

- Approach: a type system
  "if a program is typable then it doesn't go wrong"
  - The user annotates types, including the qubit allocation information
  - CNOT is typable only if there is suitable qubit connectivity

i and j must be connected in the connectivity graph Γ

$$\frac{i, j : \texttt{qidx} \in \Gamma \qquad i \rightsquigarrow j \in \Gamma}{\Gamma \vdash \texttt{cnot}[i, j] : Q(i) \otimes Q(j) \to Q(i) \otimes Q(j)}$$

i-th quantum register

Type System for Qubit Connectivity

[Wakizaka & Igarashi, 日本ソフトウェア科学会 2020]

- Typing rules. Features:
  - Target language: functional, higher-order. E.g.

$$\begin{aligned}&\texttt{let } f = \lambda j_1, j_2, j_3, j_4.\lambda\alpha.\lambda\_.\lambda g.\lambda x_1, x_2, x_3, x_4.\\ &\qquad \texttt{let } \langle x_1, x_2\rangle = g\ j_1\ j_2\ \_\ \langle x_1, x_2\rangle \texttt{ in}\\ &\qquad \texttt{let } \langle x_3, x_4\rangle = g\ j_3\ j_4\ \_\ \langle x_3, x_4\rangle \texttt{ in}\\ &\qquad \langle x_1, x_2, x_3, x_4\rangle\\ &\texttt{in}\\ &f\ i_1\ i_2\ i_3\ i_4\ ((j_1, j_2)j_1 \rightsquigarrow j_2)\ \_\ (\lambda\_.\texttt{cnot})\ q_1\ q_2\ q_3\ q_4\end{aligned}$$

  - Based on linear logic
  - CNOT requires connectivity
  - Dependent product $\Pi i.T(i)$ for qidx abstraction ("for any qidx i")
  - Qubit connectivity constraints K as types (which can further be abstracted)

$\boxed{\Gamma \vdash M : T}$ **Typing**

$$\frac{}{\Gamma \vdash () : \texttt{unit}^{\kappa}}\ \text{T-Unit} \qquad \frac{\vdash \Gamma \qquad i : \texttt{qidx} \in \Gamma}{\Gamma \vdash \texttt{meas}[i] : Q(i)^1 \to^{\omega} \texttt{bit}}\ \text{T-Meas}$$

$$\frac{\vdash \Gamma \qquad i, j : \texttt{qidx} \in \Gamma \qquad i \rightsquigarrow j \in \Gamma}{\Gamma \vdash \texttt{cnot}[i, j] : Q(i)^1 \otimes^1 Q(j)^1 \to^{\omega} Q(i)^1 \otimes^1 Q(j)^1}\ \text{T-CNot}$$

$$\frac{\vdash \Gamma \qquad x : T' \in \Gamma \qquad 1 \le |T| \qquad T \le T'}{\Gamma \vdash x : T}\ \text{T-Var} \qquad \frac{\Gamma, x : T_1 \vdash M : T_2 \qquad \kappa\Gamma = \Gamma}{\Gamma \vdash \lambda x : T_1.M : T_1 \to^{\kappa} T_2}\ \text{T-Abs}$$

$$\frac{\Gamma_1 \vdash M : T_1 \to^{\kappa} T_2 \qquad \Gamma_2 \vdash N : T_1 \qquad \vdash \Gamma_1 + \Gamma_2 \qquad \kappa \ge 1}{\Gamma_1 + \Gamma_2 \vdash MN : T_2}\ \text{T-App}$$

$$\frac{\Gamma \vdash M : T_1 \qquad \Gamma \vdash T_1 \oplus^{\kappa} T_2}{\Gamma \vdash \texttt{inl}\ M : T_1 \oplus^{\kappa} T_2}\ \text{T-SumL} \qquad \frac{\Gamma \vdash M : T_2 \qquad \Gamma \vdash T_1 \oplus^{\kappa} T_2}{\Gamma \vdash \texttt{inr}\ M : T_1 \oplus^{\kappa} T_2}\ \text{T-SumR}$$

$$\frac{\Gamma_1 \vdash P : T_1 \oplus^{\kappa} T_2 \qquad \Gamma_2, x : T_1 \vdash M : T_3 \qquad \Gamma_2, y : T_2 \vdash N : T_3 \qquad \kappa \ge 1 \qquad \vdash \Gamma_1 + \Gamma_2}{\Gamma_1 + \Gamma_2 \vdash \texttt{match}\ P\ \texttt{with}\ (x \mapsto M \mid y \mapsto N) : T_3}\ \text{T-Match}$$

$$\frac{\Gamma_1 \vdash M : T_1 \qquad \Gamma_2 \vdash N : T_2 \qquad \kappa \le \min\{|T_1|, |T_2|\} \qquad \vdash \Gamma_1 + \Gamma_2}{\Gamma_1 + \Gamma_2 \vdash \langle M, N\rangle : T_1 \otimes^{\kappa} T_2}\ \text{T-PairCons}$$

$$\frac{\Gamma_1 \vdash M : T_1 \otimes^{\kappa} T_2 \qquad \Gamma_2, x : T_1, y : T_2 \vdash N : T_3 \qquad \kappa \ge 1 \qquad \vdash \Gamma_1 + \Gamma_2}{\Gamma_1 + \Gamma_2 \vdash \texttt{let } \langle x, y\rangle = M\ \texttt{in}\ N : T_3}\ \text{T-PairDest}$$

$$\frac{\omega \cdot \Gamma_1, f : T_1 \to^{\omega} T_2 \vdash \lambda x : T_1.M : T_1 \to^{\omega} T_2 \qquad \Gamma_2, f : T_1 \to^{\omega} T_2 \vdash N : T_3 \qquad \vdash \omega \cdot \Gamma_1 + \Gamma_2}{\omega \cdot \Gamma_1 + \Gamma_2 \vdash \texttt{let rec } f = \lambda x : T_1.M\ \texttt{in}\ N : T_3}\ \text{T-Rec}$$

$$\frac{\Gamma, K \vdash M : T}{\Gamma \vdash \lambda\_ : K.M : K \Rightarrow T}\ \text{T-Conn} \qquad \frac{\Gamma \vdash M : K \Rightarrow T \qquad K \subseteq \Gamma}{\Gamma \vdash M\_ : T}\ \text{T-ConnApp}$$

$$\frac{\Gamma, i : \texttt{qidx} \vdash M : T}{\Gamma \vdash \lambda i : \texttt{qidx}.M : \Pi i : \texttt{qidx}.T}\ \text{T-QAbs} \qquad \frac{\Gamma \vdash M : \Pi i : \texttt{qidx}.T \qquad j : \texttt{qidx} \in \Gamma}{\Gamma \vdash M\ j : [j/i]T}\ \text{T-QApp}$$

$$\frac{\Gamma, \alpha : \texttt{graph}^{(n)} \vdash M : T}{\Gamma \vdash \lambda\alpha : \texttt{graph}^{(n)}.M : \Pi\alpha : \texttt{graph}^{(n)}.T}\ \text{T-GAbs}$$

$$\frac{\Gamma \vdash M : \Pi\alpha : \texttt{graph}^{(n)}.T \qquad \Gamma, i_1 : \texttt{qidx}, \ldots, i_n : \texttt{qidx} \vdash K}{\Gamma \vdash M\ (i_1, \ldots, i_n)K : [(i_1, \ldots, i_n)K/\alpha]T}\ \text{T-GApp}$$

Type System for Qubit Connectivity
[Wakizaka & Igarashi, 日本ソフトウェア科学会 2020]

- Correctness theorems　“if a program is typable then it doesn’t go wrong”
  - **Thm.** (Progress) If a program M is typable ($C \vdash [\psi, M] : T$), then

    M is either a value, or M has a reduction $C \vdash [\psi, M] \to_p [\psi', M']$

Special programs for output of computation (constants, functions, ...)

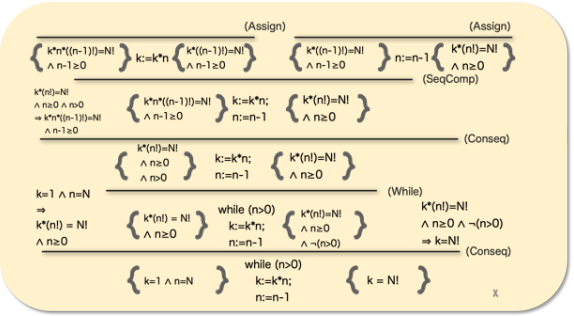
A computation step in λ-calculus

  - **Thm.** (Preservation)

Types are preserved along reduction

$C \vdash [\psi, M] : T$ and $[\psi, M] \to_p [\psi', M']$ imply $C \vdash [\psi', M'] : T$.

Cf. proof tree in Hoare logic



Type System for Qubit Connectivity
[Wakizaka & Igarashi, 日本ソフトウェア科学会 2020]

- Usage (note that this is **static**)
  - A programmer writes a program M with type annotation (amounts to writing a type derivation tree)
  - Type checker returns no type error (i.e. all derivation steps adhere to rules)
    ➔ the programmer is sure that there will be no runtime error that is due to qubit connectivity

Type derivation tree

$$C \vdash [\psi, M] : T$$

- Future work
  - Type checking (“does the user-specified qubit allocation respect connectivity?”)
    ➔ type inference
    (automatic search for type derivation trees, i.e. automated qubit allocation)
  - Accommodating qubit reallocation (costly but sometimes inevitable)

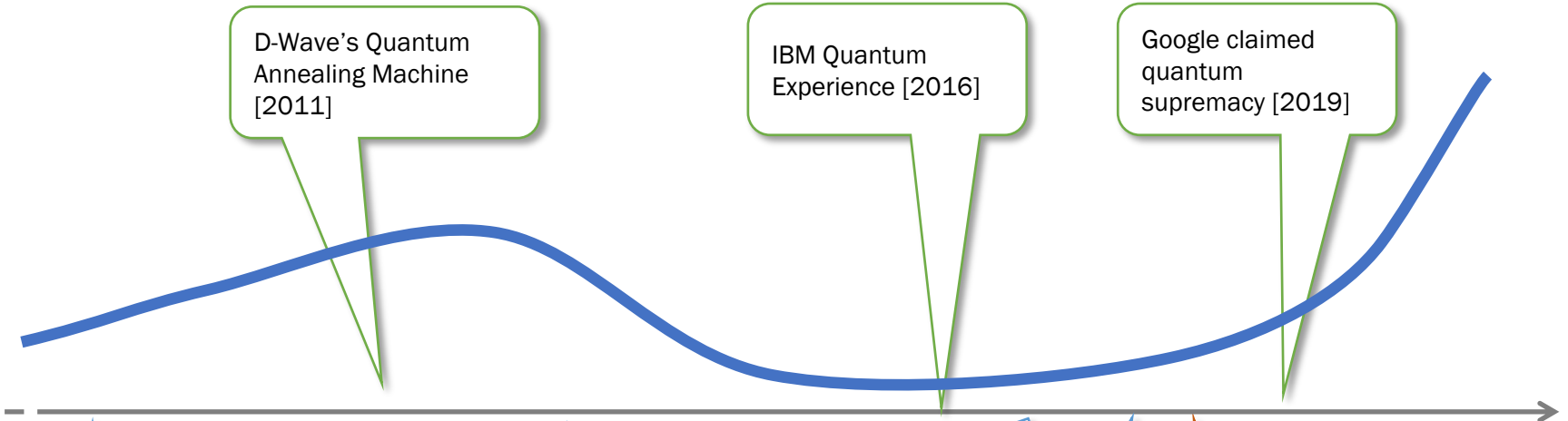
Outline

- Introduction
- Hoare logic as a showcase of static analysis
- QPL Work (1): Ying's quantum Hoare logic
- QPL Work (2): Quantum GoI for semantics of quantum λ-calculus
  [H. & Hoshino]
- "Static analysis for hardware-specific, low-level concerns in NISQ"
- QPL Work (3): Quantum Hoare logic for bounding errors [Hung+, POPL'19]
- QPL Work (4): Language-Based Automated Uncomputation [Bichsel+, PLDI'20]
- QPL Work (5): Type System for Qubit Connectivity
  [Wakizaka & Igarashi, JSSST Congress '20]
- Towards Quantum IDE

Trends in Quantum Programming Language (QPL) Research

repeat

Not at all comprehensive!



QCL
[Ömer 2000]
First QPL, imperative

quantum λ-calculus
[van Tonder, SIAM J. Comp. 2004]
Functional, linear types for no-cloning,

Quantum lambda calculus [Selinger & Valiron, 2008, 2009]
Functional, categorical semantics

Quantum I/O monad
[Green & Altenkirch, 2009]
Making Haskell a QPL

Quantum flowchart language
[Selinger, MSCS 2004]
Formal semantics by superoperators

Quantum lambda calculus [Hasuo & Hoshino, LICS 2011]
Categorical GoI semantics for "quantum data, classical control"

Quantum Hoare Logic
[Ying, ACM TOPLAS 2012]
Program logic for correctness of imperative quantum programs

Quipper
[Green, Lumsdaine, Ross, Selinger, Valion, PLDI 2013]
A functional language that generates quantum circuits

Quantum lambda calculus
[Pagani, Selinger, Valion, POPL 2014]
Another categorical semantics

QWIRE
[Paykin, Rand, Zdancewic, POPL 2017]
Functional, generates circuits, Coq-integrated

Q#
[Svore et al., RWDSL 2018]
by Microsoft, functional/imperative

Logic for qubit error analysis
[Hung, Hietala, Zhu, Ying, Hicks, Wu, POPL 2019]
Quantum Hoare logic adapted for bounding errors

Silq
[Bichsel, Baader, Gehr, Vechev, PLDI 2020]
Functional, automatic uncomputation & a type system for it

Type system for qubit connectivity
[Wakizaka & Igarashi, JSSST Congress 2020]
Type system for Selinger's quantum λ-calculus that ensures qubit connectivity

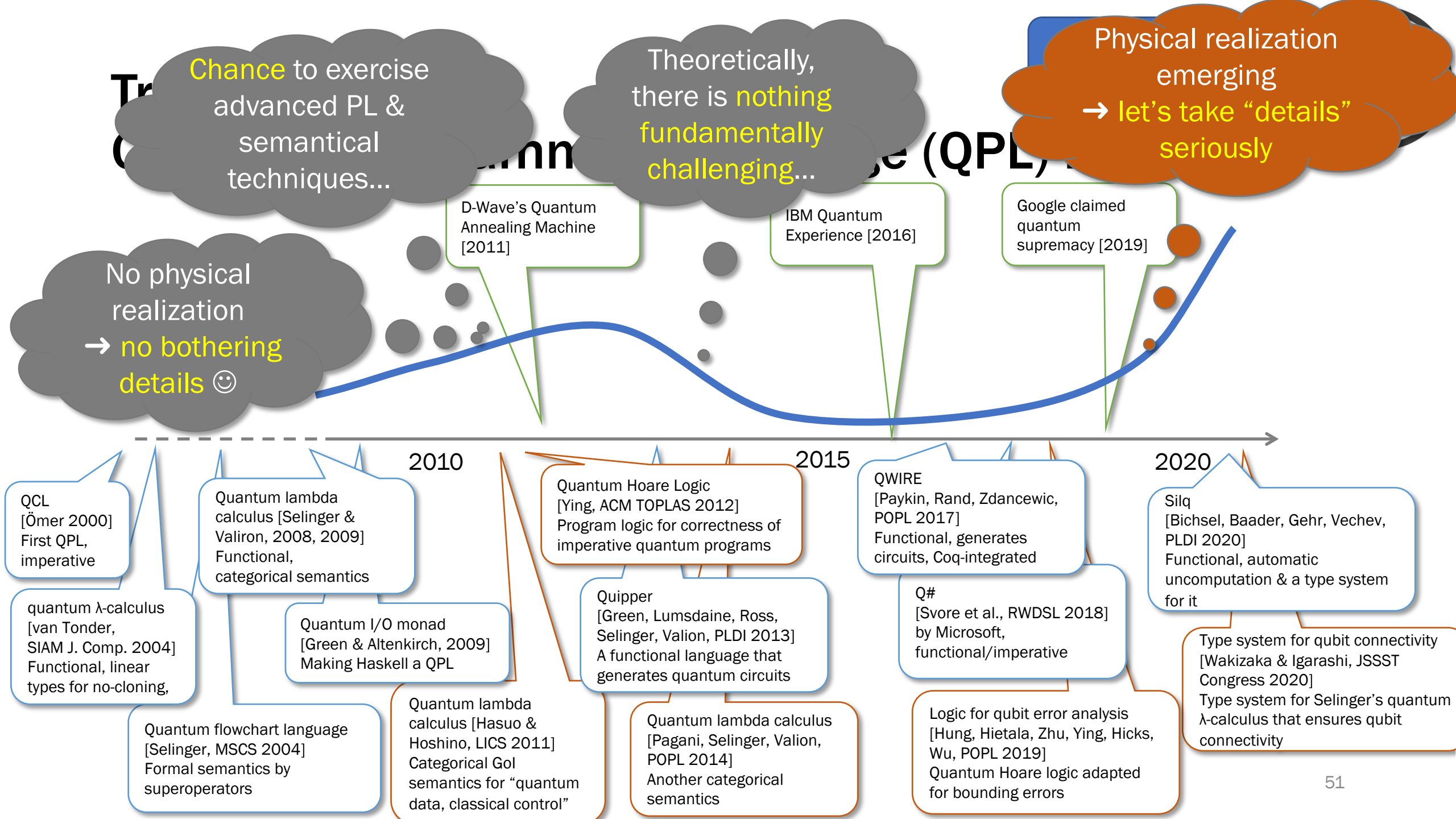
Tr… (QPL)…

Chance to exercise advanced PL & semantical techniques…

Theoretically, there is nothing fundamentally challenging…

Physical realization emerging → let's take "details" seriously

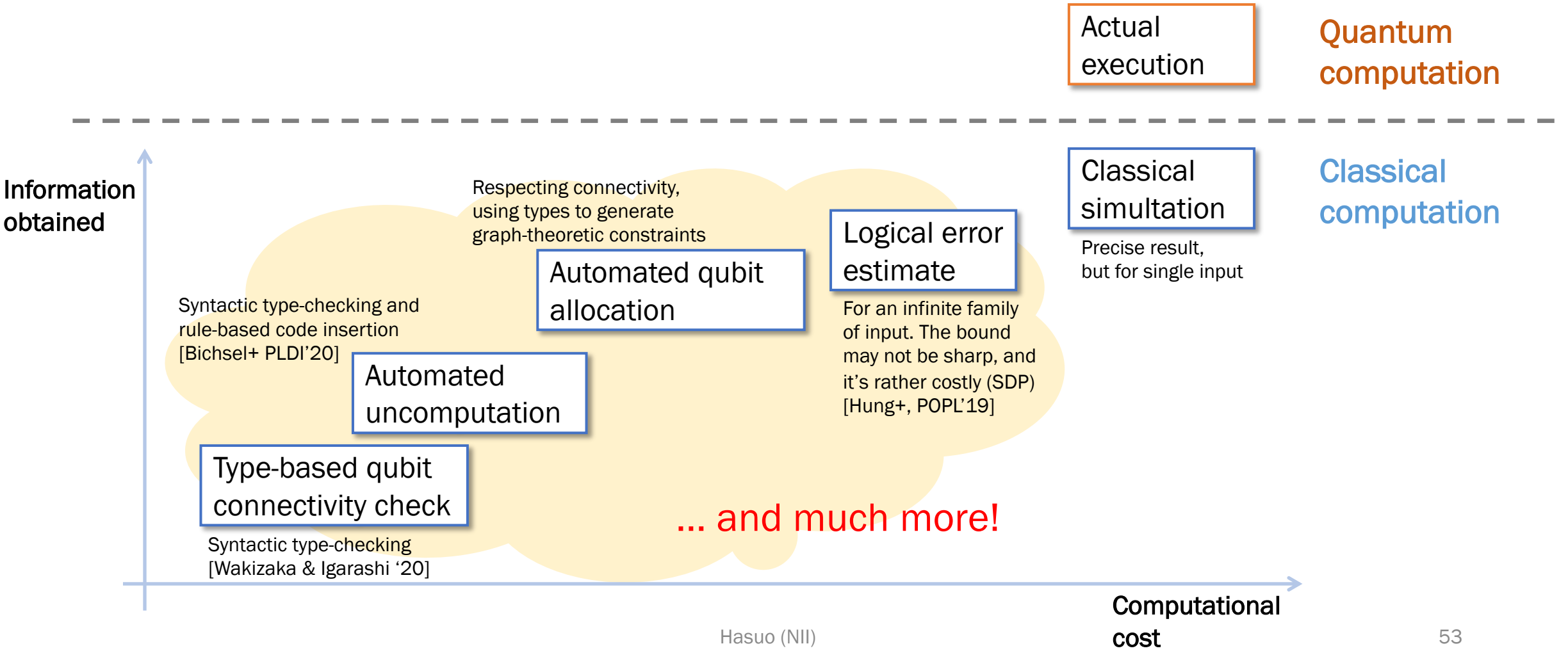
No physical realization → no bothering details ☺



Static Analysis in the Quantum Age

- Static analysis: reasoning about programs without execution
  - Goals, conventionally:
    correctness (in Hoare logic, …),
    security (non-interference [Sabelfeld & Myers '03], differential privacy [Barthe+ '14], …),
    complexity analysis [Hofmann'00], …
- For quantum programs?
  - Execution is costly and probabilistic
    → static analysis makes even bigger sense
  - Can be seen as generalization of classical simulation
- "Static analysis for hardware-specific, low-level concerns in NISQ"
  - Error estimate [Hung+, POPL'19]
  - Automated uncomputation [Bichsel+, PLDI'20]
  - Qubit connectivity [Wakizaka & Igarashi, JSSST Congress '20]

Static Analysis and Quantum Software



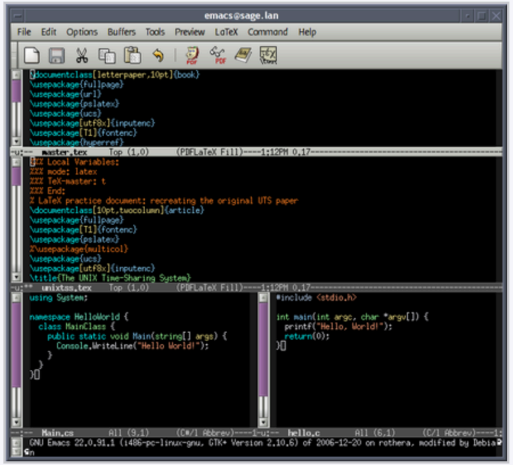
Towards Quantum IDE (Integrated Development Environment)

- Support quantum programmers to write correct and efficient code
  - Conventional IDE functionalities:
    syntax highlighting, code completion, refactoring, version control,
    debugging support, code search, …

For managing a big codebase…
- Not relevant to quantum programs (yet)
- Works for quantum programs as well

  - + quantum-specific functionalities:
    error estimate, automated uncomputation, automated qubit allocation, …
    - More semantical consideration is needed
      ➜ static analysis, logical and type-theoretic techniques from the PL community
    - Spare quantum trials & errors by classical static reasoning
  - Choice/design of a QPL is important
    (structured programming [Dijkstra] ➜ better logical support)

wikipedia

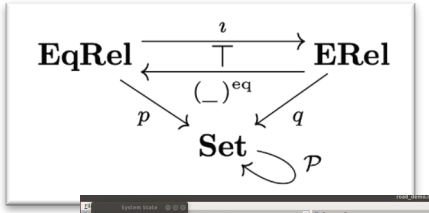


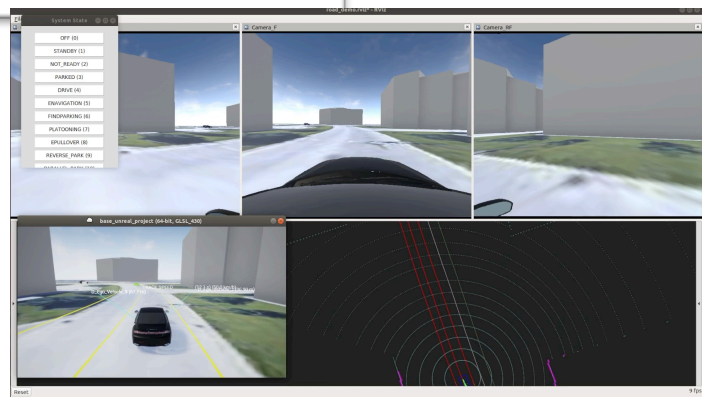
GNU Emacs, an extensible editor that is commonly used as an IDE on Unix-like systems

Towards Quantum IDE (Integrated Development Environment)

- To the PL community:
  it's time to "get our hands dirty"
  - Researchers prefer fundamental challenges
    (logical correctness, soundness and completeness meta-theorems, …)
  - In comparison, low-level concerns in NISQ may seem to be minor
  - but they can really benefit from static analysis techniques
- Theoreticians diving into applications
  - Need to dive in
    — nobody formulates a clean theoretical problem for you
  - Matching practical needs and theoretical seeds
    — fun and rewarding
  - (in our ERATO project:
    category theorists in automated driving safety)

Dynamic analysis (testing) is costly!



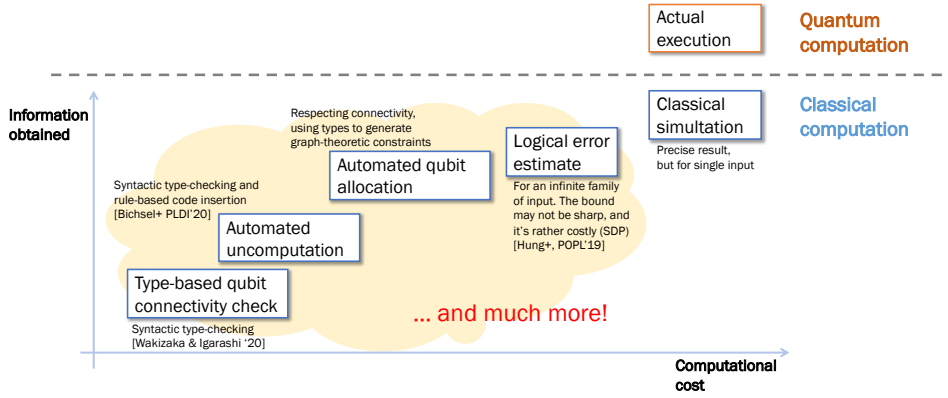


Summary

ご清聴ありがとうございました
蓮尾 一郎（NII, i.hasuo@acm.org）
In collaboration with
星野 直彦（崇城大），脇坂 遼・五十嵐 淳 （京大）

ERATO MMSD

- Static analysis: reasoning about programs without executing it
- "Static analysis for hardware-specific, low-level concerns in NISQ"
  - Execution is costly and probabilistic ➔ dynamic analysis (testing) is hard
  - Hardware limitation (NISQ) ➔ a lot of low-level concerns
- Examples:
  logical error bounds [Hung+ '19],
  automatic uncomputation [Bichsel+ '20],
  connectivity check [Wakizaka & Igarashi '20], …



- Quantum IDE for correct and efficient code, with the help of static analysis
- To PL people: QPL is a gold mine. Let's dive in!
  To others: involve (open-minded) PL people in your quantum software project!