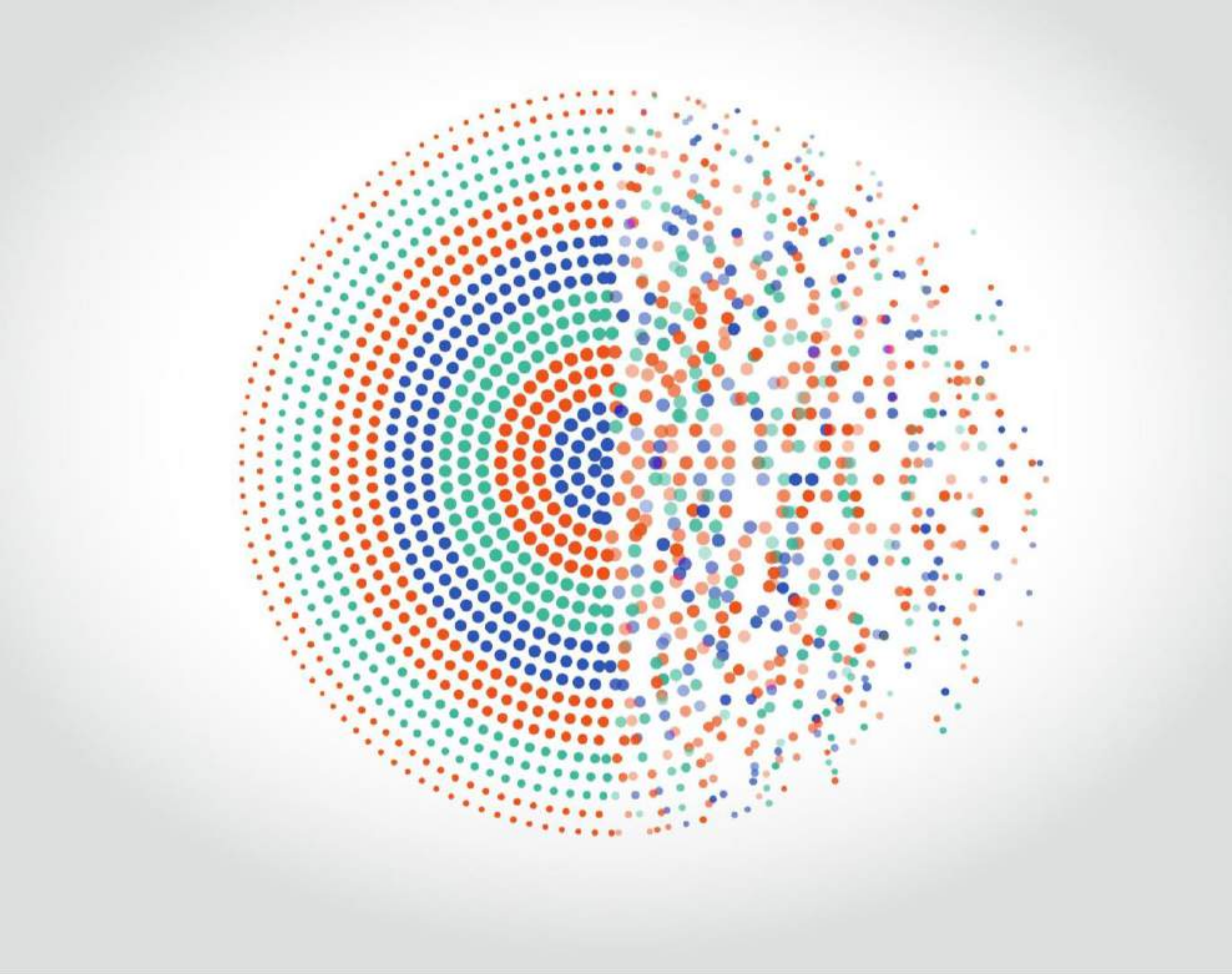
Formal Modelling, Programming & Verification of Quantum Systems

Richard Bornat
Rajagopal Nagarajan

Middlesex University London

BCS FACS October 19, 2021



Motivation

- Formal Verification for reasoning about correctness and security of classical systems is used routinely by Microsoft, Intel, NASA, Amazon, Facebook, etc.
- Can we do something similar for Quantum Computing and Quantum Cryptography?
- General purpose, large-scale, Quantum Computers some years away. RSA not believed to be under threat at the moment.
- Big push recently by companies such as IBM, Google, Intel, Honeywell for “quantum supremacy”.

NISQ Computers

IBM Q

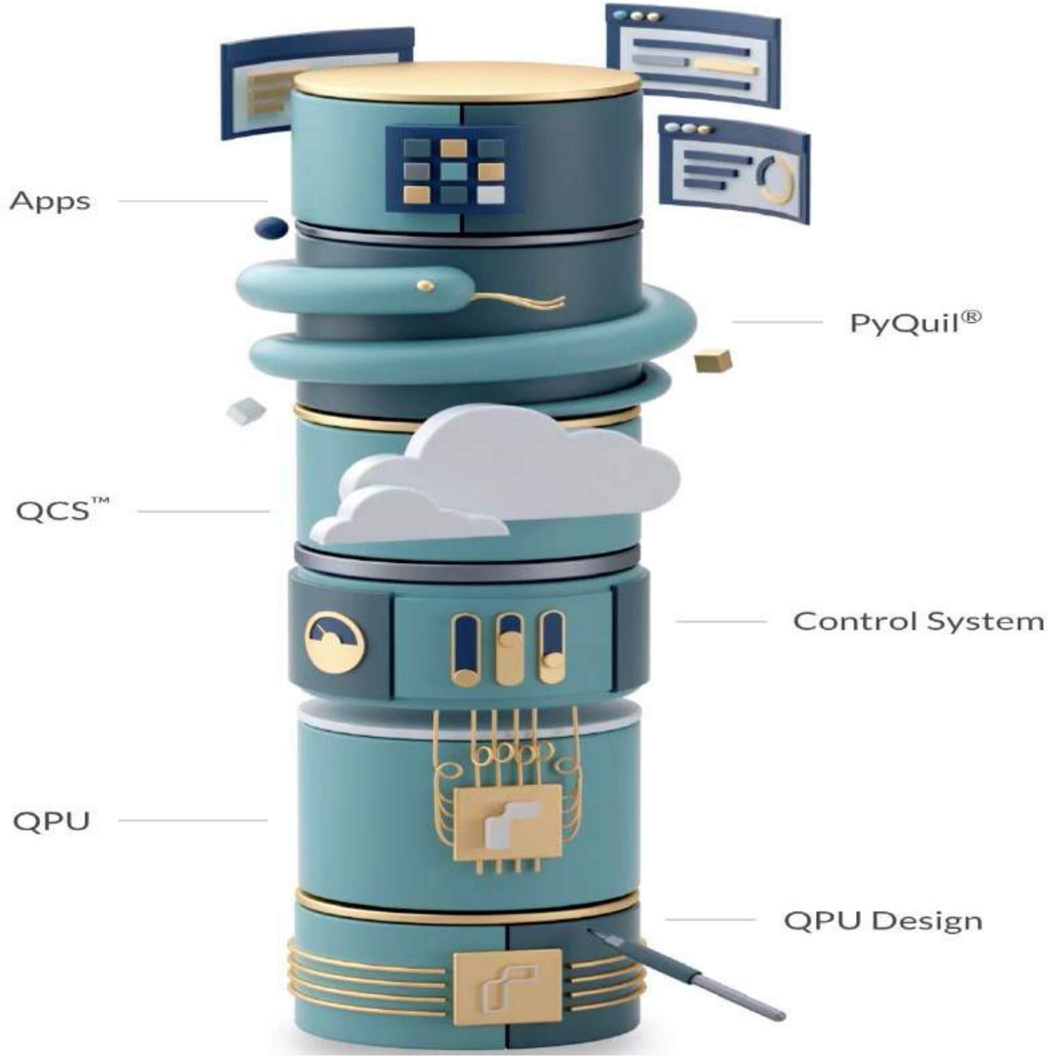
NISQ Computers

Google Bristlecone

Rigetti

- Small number of quantum bits (50—75)
- Noisy, prone to errors
- Emerging applications to quantum molecular simulation, quantum machine learning, optimisation

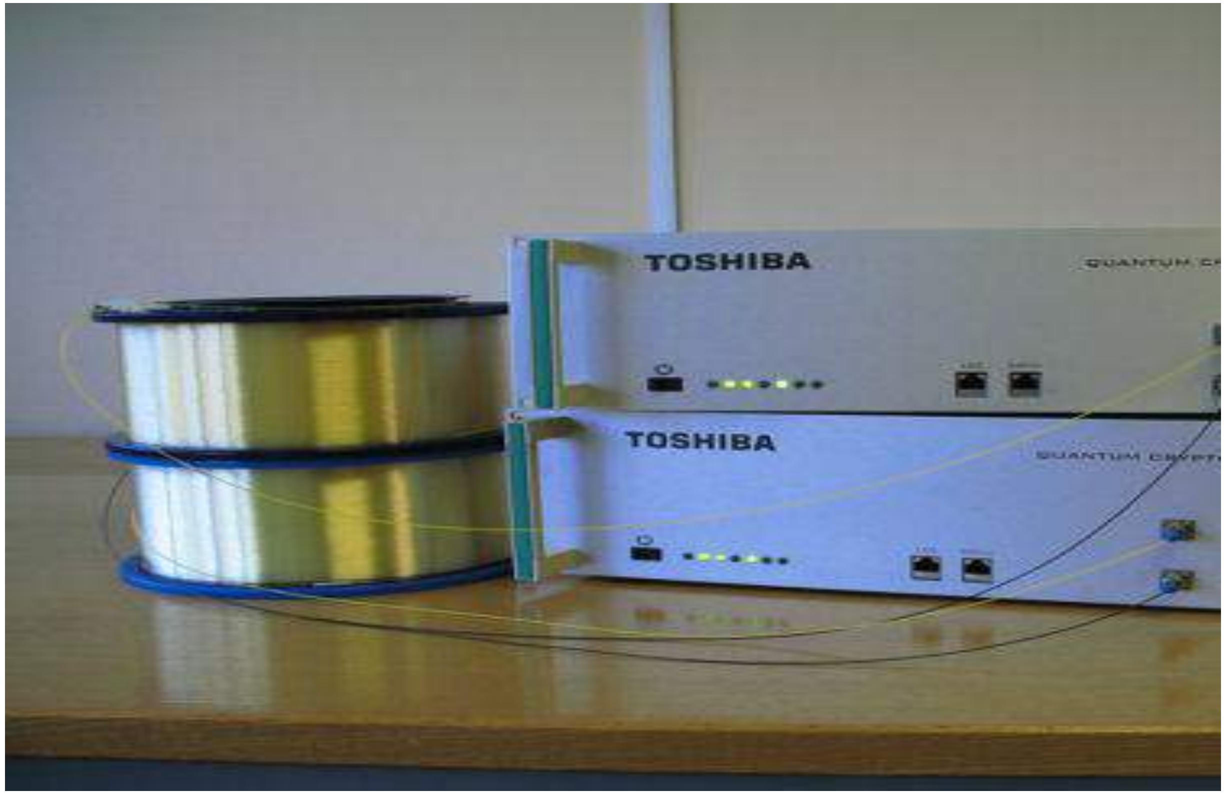
Full Stack (Rigetti)



- Each component as well as the whole system need to work correctly.

Quantum Communication and Cryptography

- Quantum Communication and Cryptography mature field.
- Sending secret keys encoded in photons; eavesdropper will be detected. Not much computation.
- QKD unconditionally secure. How about implementations?



SeCoQC QKD Network

UK QKD Network

- Real-world experience for well over a year.
- Commercial id Quantique QKD equipment.
- Installation over BT fibre optic cables and through BT exchanges.

Other QKD Networks

- Recent announcement by BT and Toshiba about a metro QKD network across London
- China launched a satellite Micius for secure communication using QKD.
- BT interested in testing and formal verification.

Qubits

Given basis states $|0\rangle$ and $|1\rangle$, the state of a quantum system is given by a linear combination of the two (superposition):

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

where α and β are complex numbers with $|\alpha|^2 + |\beta|^2 = 1$.

Example: $\sqrt{0.3}\,|0\rangle + \sqrt{0.7}\,\,|1\rangle$

Measurement

A qubit or quantum state in superposition

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

when measured (or observed) collapses to a classical state $|0\rangle$ with probability $|\alpha|^2$ or the state $|1\rangle$ with probability $|\beta|^2$.

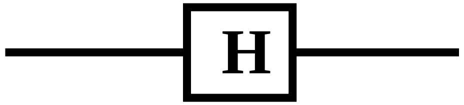
If you measure $\sqrt{0.3}\,|0\rangle + \sqrt{0.7}\,|1\rangle$, you get $|0\rangle$ with probability 0.3 and $|1\rangle$ with probability 0.7.

The Hadamard Gate

- The Hadamard gate acts on one qubit, and places it in an equal superposition of $|\mathbf{0}\rangle$ and $|1\rangle$

$$\mathrm{H}|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |\mathbf{1}\rangle)$$

$$\mathrm{H}|\mathbf{1}\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |\mathbf{1}\rangle)$$



The Pauli gates

- The Pauli gates act on one qubit, as follows:

  - bit flip, **X**:

  $\mathbf{X}(\alpha\,|0\rangle + \beta\,|1\rangle) = \alpha\,|1\rangle + \beta\,|0\rangle$

  - phase shift, **Z**:

  $\mathbf{Z}(\alpha\,|0\rangle + \beta\,|1\rangle) = \alpha\,|0\rangle - \beta\,|1\rangle$

  - phase shift and bit flip, **Y**:

  $\mathbf{Y}(\alpha\,|0\rangle + \beta\,|1\rangle) = \alpha\,|1\rangle - \beta\,|0\rangle$

  - identity, **I**, does not change the input

The Controlled Not Gate

- The **CNot** gate acts on **two** qubits:

CNot($|00\rangle$ **) =** $|00\rangle$

CNot($|01\rangle$ **) =** $|01\rangle$

CNot($|10\rangle$ **) =** $|11\rangle$

CNot($|11\rangle$ **) =** $|10\rangle$

Entanglement

- Unlike classical states, there exist two-qubit quantum states that cannot be decomposed into a combination of two single-qubit states.

  Example:

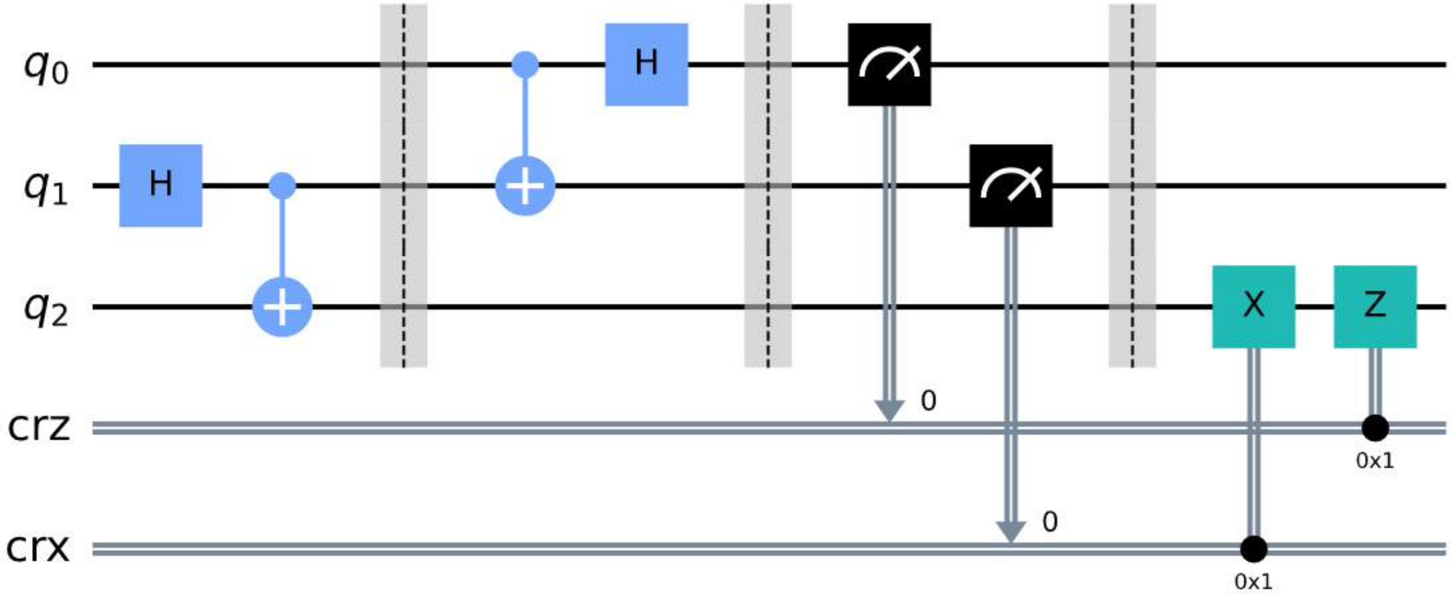
$$\frac{1}{\sqrt{2}}(|0\mathbf{0}\rangle + |\mathbf{11}\rangle)$$

- Measuring one qubit always fixes the state of the other instantaneously, even though they might be some distance apart.

Quantum Teleportation

- Sending an unknown qubit from Alice to Bob without a quantum channel.
- Alice and Bob share prior entanglement.
- They also have a classical channel for communication
- The original qubit is destroyed.

Quantum Teleportation on IBM Q



Quantum Teleportation on IBM Q

File Edit View Run Kernel Tabs Settings Help

teleportation.ipynb

Markdown   Qiskit v0.30.1 (ipykernel)

5.2 Executing

[28]:
```
# First, see what devices we are allowed to use by loading our saved accounts
IBMQ.load_account()
provider = IBMQ.get_provider(hub='ibm-q')
```

```
ibmqfactory.load_account:WARNING:2021-10-16 20:27:32,466: Credentials are already in use. The existing
account in the session will be replaced.
```

[29]:
```
# get the least-busy backend at IBM and run the quantum circuit there
from qiskit.providers.ibmq import least_busy
from qiskit.tools.monitor import job_monitor
backend = least_busy(provider.backends(filters=lambda b: b.configuration().n_qubits >= 3 and
                                   not b.configuration().simulator and b.status().operational==True))
t_qc = transpile(qc, backend, optimization_level=3)
job = backend.run(t_qc)
job_monitor(job)  # displays job status under cell
```

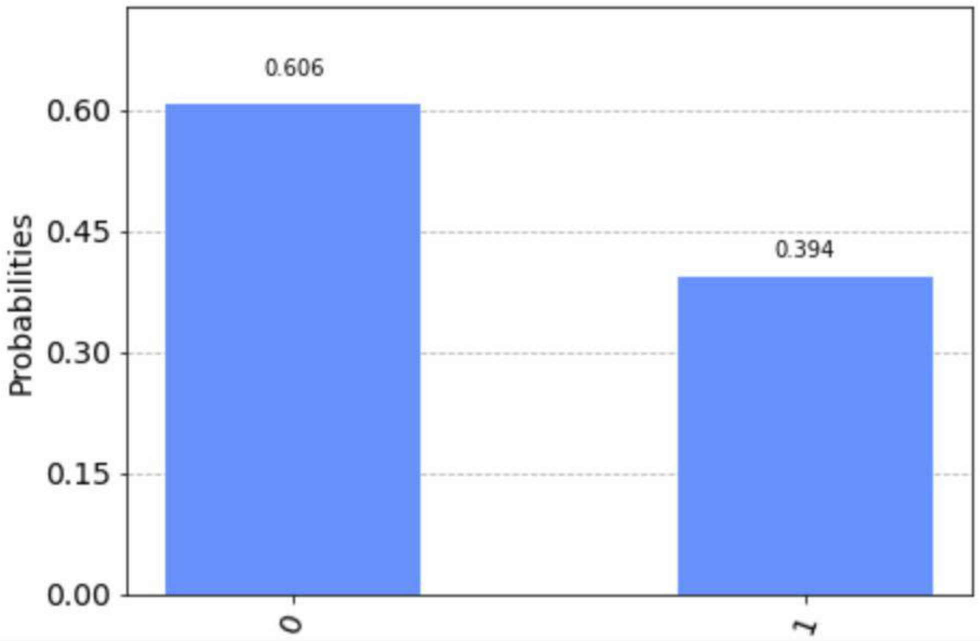
```
Job Status: job has successfully run
```

Quantum Teleportation on IBM Q

```
[23]: # Get the results and display them
exp_result = job.result()
exp_counts = exp_result.get_counts(qc)
print(exp_counts)
plot_histogram(exp_counts)
```

```
{'0': 621, '1': 403}
```

[23]:



Python is the FORTRAN of Quantum Programming

- Need special purpose programming languages
- Enables type-checking
- Reasoning about program correctness
- Communicating Quantum Processes (CQP), based on pi-calculus. Linear types enforce no-cloning.
- Published in POPL 2005, MSCS journal
- Similar efforts: QPL/Quipper, Microsoft Q# (not for distributed computation)

Introduction to Formal Verification

- Specification - What is a system supposed to do?
- Verification - Does the system do what it is supposed to do?
- "Formal Verification" is the act of **proving** or **disproving** the correctness of intended algorithms underlying a system with respect to a certain **formal specification** or property, using formal **mathematics**
- Algorithms (software) for checking if the system **satisfies** the specification

The Failure of the Arianne 5 Rocket

Why did the Arianne 5 Fail?

There were two main reasons behind the failure of the rocket:

- Software failure occurred when an attempt to convert a 64 bit floating point number to a 16 bit signed integer failed due to overflow and raised an exception
  - There was no exception handling for this and so the system exception handling routines were invoked which shut down the system
- Inertial reference system failed and the system backup shutdown
  - Diagnostic commands were sent to the engine which interpreted them as real commands

Failure of the Patriot Missiles

- The missile system failed to track and target an incoming Scud missile
- The problem in the missiles tracked to “accumulating linear error of .003433 seconds per 1 hour of uptime”
- This caused 28 US soldiers to lose their lives

Other Examples of Software errors

- Therac – 25 (1985-87): A computer-controlled radiation therapy system massively overdosed 6 people

- Pentium – FDIV bug (1994): Mistake in the implementation of division algorithm in Pentium, leading to incorrect answers in some situations at or beyond 4 digits, this cost them around $475 million

- “Program **testing** can be used to show the **presence** of **bugs**, but never to show their absence!” ~Edsger W. Dijkstra

Successes of Formal Verification

Formal Verification has been successfully applied in numerous cases

- The CompCert project investigated the formal verification of realistic compilers for critical embedded software. The main result of this project was the CompCert C verified compiler, a high assurance compiler for almost all of the C language.
- The Paris Metro line 14 employed a formal verification method called B-Method in order to automate processes.
- Routinely used by large companies.

Formal Verification Techniques

Model Checking:
This state-based method involves analysing the properties of a model of the proposed system. This algorithm can provide a counter-example if a property is not satisfied.

Theorem Proving:
Theorem-proving involves the creation of a mathematical model for the system as definitions in mathematical logic. It then derives the properties of system as proofs from the mathematical definition.

Equivalence Checking:
Equivalence Checking is a method of formal verification which attempts to verify whether two systems (hardware or software) are functionally the same.

Model Checking

- This is a method of automated verification.

- It consists in **mechanically proving that a model**, σ, expressed in a suitable modelling language, **satisfies a (temporal) logic formula** ϕ, written $\sigma \models \phi$. Otherwise a counterexample can be produced.

- **Gavin Lowe** used a model checker to detect a subtle security flaw in the Needham Schroeder public key protocol.

- Classical **security protocols** are frequently verified using model checking.

Analysis of Quantum Cryptographic Protocols and Systems

- Protocols for quantum key distribution are ideal targets for verification
  - Possible detection of subtle flaws (cf. Needham-Schröder PKCS protocol)
- Availability of commercial QKD systems and networks
  - Need for tools for validating implementations
  - Verification of classical pre- and post-processing procedures
  - Verification of classical hardware and interface components

Model-checking Challenges

- In order to perform model-checking of quantum protocols, we need to consider the following issues:
  - The state space of a single qubit is **infinite**
  - The state space of an *n*–qubit system grows **exponentially** with *n*
  - There are infinitely many possible quantum operators
  - Quantum measurement is **probabilistic**

Quantum Model Checker

- Initial work used a probabilistic model checker (PRISM) to analyse the BB84 QKD protocol.
- We can compute, for example, the probability of detecting an eavesdropper when N qubits are transmitted; and the probability that the eavesdropper obtains certain number of transmitted bit values.
- QMC is a verification tool comprising:
  - A typed, concurrent specification language for quantum protocols
  - A polynomial-time simulator for quantum computations involving Clifford operators on stabilizer states
  - An evaluator for the logic QCTL (quantum computation tree logic) [Chadha, Mateus,… 2006]

Quantum Model Checker

Step 1: System Model
Specify protocol behaviour using a modelling language, eg: QMCLang

Step 2: Property Description
Describe protocol properties (desirable & undesirable behaviour)
logic: EQPL/QCTL

Step 3: Verification
Pass the model and properties into model-checking tool which will check whether

$$|\psi\rangle, c \models \phi$$

Quantum Model Checker

- The logic QCTL [Baltazar, Chadha, Mateus 08] is a CTL variant.
- built atop quantum propositional logic (EQPL) [Mateus & Sernadas 06].
- QCTL allows us to reason about properties of quantum state.
- We can check whether two states are entangled, for example.

Publications

Proposed the application of Formal Verification to Quantum Systems about 19 years ago.

- Model-checking (CAV ‘08, CUP Book Chapter), with Simon Gay and Nikolaos Papanikolaou
- Equivalence checking (TACAS’13, TACAS ‘14, ACM ToCL), with Simon Gay and Ebrahim Ardeshir-Larijani.
- Theorem proving using Coq (QIP ‘14 poster, QPL ‘15), with Jaap Boender and Florian Kammueller.
- qtpi, implementation with a symbolic simulator (TACAS ‘20, ICoQC ‘18, AQIS ’19 poster), with Richard Bornat and others.
- Property-based Testing (QSE ‘20), with Mohammad Mousavi and Shahin Honarvar.

Software Tools

- QMC: A Quantum Model Checker.
- QEC: A Quantum Equivalence Checker. (http://www.dcs.gla.ac.uk/~simon/qec/).
- Qtpi: A symbolic simulator for CQP. (https://github.com/mdxtoc/qtpi).
- QSharpCheck: Property-based testing of Q# programs. (https://github.com/ShahinHonarvar/QSharpCheck).