Model Checking Recursive Quantum Protocols

Linda Anticoli

Dept. of Mathematics, Computer Science and Physics - University of Udine, Italy.
School of Computing Science - Newcastle University, UK.

June 1996
Ariane 5 launcher failure

"Loss of information due to specification and design errors in the software of the inertial reference system."

Motivations

Quantum information is more fragile than classical one

$$\Downarrow$$

flaws in the design of quantum protocols and noise in their physical implementation

Motivations

State of the art:

- Formal, higher–level specification of quantum algorithms
  Quantum Programming Languages

  - J. W. Sanders and P. Zuliani. "Quantum Programming" (2000)
  - A. van Tonder. "A lambda calculus for quantum computation" (2003)
  - A.S. Green, , et Al. "Quipper: A Scalable Quantum Programming Language" (2013).

- Formal verification of quantum algorithms
  Quantum Model Checkers

  - P. Mateus, et Al. "Towards model-checking quantum security protocols" (2007)
  - Y. Feng, et Al. "QPMC: A Model Checker for Quantum Programs and Protocols" (2015)

Motivations

Desiderata:

High-level formalisms allowing to define and automatically verify formal properties of algorithms abstracting away from low–level physical details:

- L. Anticoli, et Al. "Towards quantum verification: From Quipper circuits to QPMC" (2016)
- L. Anticoli,et Al. "Entangλe: A Translation Framework from Quipper Programs to Quantum Markov Chains" (2017).

Preliminaries and Notation

Question 1

What is a quantum algorithm (or quantum protocol)?

$$\Downarrow$$

Quantum Computation and Information

Question 2

What does model–checking mean?

$$\Downarrow$$

Formal Methods in Computer Science

Quantum Computation and Information – Remarks

Paradigm of computation concerned with computational tasks, and information processing achieved through quantum mechanical systems.

Efficient solutions for classically hard problems

- Integer Factoring $n = log_2 N$
  - Classical Solution -> $\approx \exp[O(n^{1/3} log^{2/3} n)]$;
  - Shor's Algorithm -> $\approx O(n^3)$;
- Unsorted Database Search
  - Classical Solution -> $O(N)$;
  - Grover's Algorithm -> $O(N^{1/2})$;

Quantum Computation and Information – Remarks

Efficiency

- **Parallelism**: linearity of space and operators;
- **Interference**: the states *interfere* deleting the "wrong" ones, while increasing the probability of the desired outcome.
- **Correlations**: non–local correlations between the outcomes of measurements performed on different qubit strings.

Qubit

Superposition of States

Quantum analogue of a classical bit. State of a 2-level system:

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

where $|\alpha|^2 + |\beta|^2 = 1$ and $\alpha, \beta \in \mathbb{C}$

Quantum Register

Quantum analogue of a classical bit string composed by n–qubits:

$$|\psi_{tot}\rangle = |\psi_1\rangle \otimes \cdots \otimes |\psi_n\rangle$$

allowing 2^n superposed basis states.

Quantum Circuit Model

Quantum Gates

Quantum counterpart of classical logic gates.

n qubits $\longrightarrow$ quantum gates: $2^n \times 2^n$ unitary operators.

Single Qubit Gates

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Controlled Gates

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

tipically used to create correlations.

Quantum Circuits

Quantum algorithms are represented by quantum circuits in which the computation is realised by the following steps.

1. State preparation;
2. Application of unitary operators;
3. Measurement.

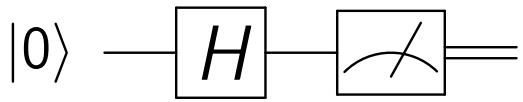


Figure: Quantum Coin Tossing Circuit.

Applications

Quantum Teleportation

Quantum information (qubits) is transmitted from a location to another by means of classical communication and previously shared entangled couples between sender and receiver.

Quantum Cryptography

Use of quantum effects to perform cryptographic tasks.
Measurement disturbs the data –> eavesdropper can be detected!

Refs:

C. H. Bennett, et Al. *"Teleporting an Unknown Quantum State via Dual Classical and Einstein-Podolsky-Rosen Channels"*

C. H. Bennett, et Al. *"Quantum cryptography: Public key distribution and coin tossing"*

Issues

Problem 1

Quantum circuits are low level descriptions of computation.

Problem 2

Quantum circuits are not Turing complete, no recursion.

Quantum Programming Languages

Solution1

Quantum Programming Languages: abstract the computation from the physical low–level detail to a human readable, and formally defined *high–level* description.

- **Quantum pseudocode**
- **QCL**
- **Q Language**
- **qGCL**
- **Quantum Lambda Calculus**
- **Quipper**
- **LIQUi$|\rangle$**
- ...

Quipper

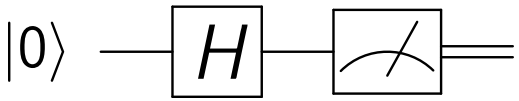
Functional programming for quantum computation.

- Based on Haskell;
- The semantics of a Quipper program is given in terms of extended quantum circuits;
- Allows to generate a graphical representation of the implemented circuit, but not of quantum programs;
- Provides three different of simulators.

A Quipper program is a function that inputs some quantum and classical data, performs state changes on it, and then outputs the changed quantum/classical data.

Quipper example

```
qCoinFlip :: Qubit -> Circ Bit
qCoinFlip q = do
  q <- qinit False
  hadamard_at q
  c <- measure q
  return c
```




```
qCoinFlipRec :: Qubit -> recCirc ()
qCoinFlipRec q = do
  q <- qinit False
  hadamard_at q
  c <- measure q
  [...]
  if c==0 then
    return c
  else
    return qCoinFlip(q)
```

Quantum Markov Chains

Solution 2

Data–structures allowing to model recursion in quantum algorithms: Quantum Markov Chains.

Quantum Markov Chain

Tuple (S, Q, AP, L), where:

- S is a countable (finite) set of classical states;
- $Q : S \times S \to S^{\mathcal{I}}(\mathcal{H})$ is the transition matrix where for each $s \in S$,the operator $\sum_{t \in S} Q(s, t)$ is trace-preserving;
- AP is a finite set of atomic propositions;
- $L : S \to 2^{AP}$ is a labelling function.

Example

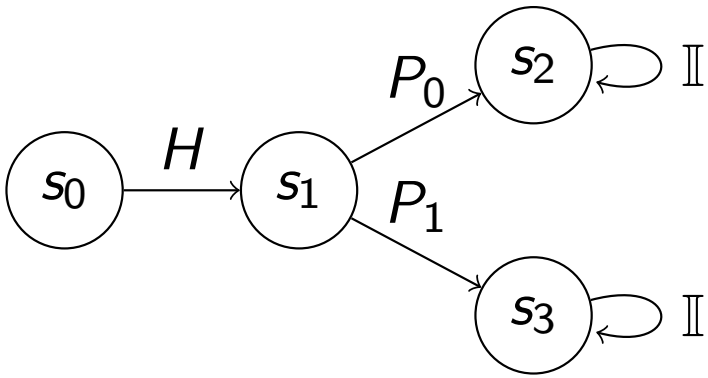


Figure: QMC for Quantum Coin Tossing.

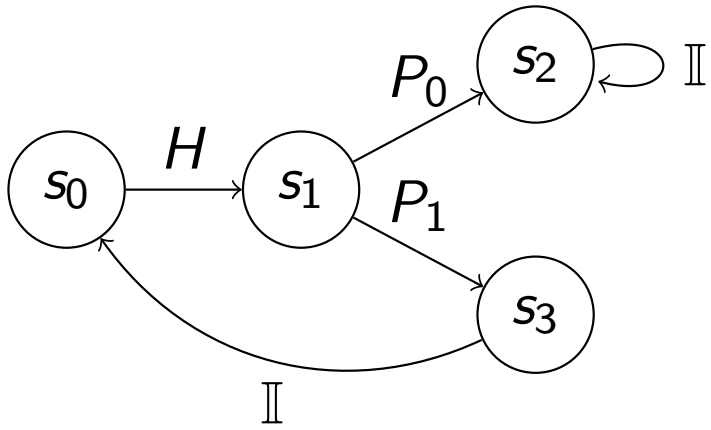


Figure: QMC for Recursive Quantum Coin Tossing.

QMCs are more expressive! So, let's use them.

Equivalent Behaviour

Bisimilarity

A quantum circuit can always be translated in a QMC with the same behaviour, while the converse is not possible.

(Boring proofs in references)

Now what?

- Formal, high–level language to express quantum computations ☑
- Formal definition of recursion in quantum programs ☑
- Formal verification of quantum programs ☐

Quantum Model Checking (1)

Model Checking

Exhaustive exploration of the state space of a system to verify (or falsify) if a temporal property is satisfied.

Step–by–step

- Abstract model of the system;
- Temporal logic to specify the properties.

Quantum Model Checking (2)

Abstract Model

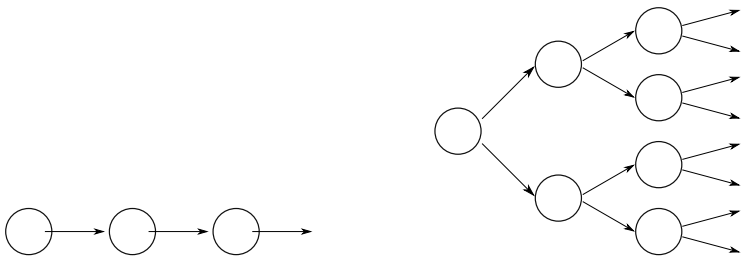
Graph structure representing the computation steps. Classically: Kripke structures, LTS, DTMC.
QMC can be used as a model for quantum computation!

Temporal Logics

Modal logics used to express time–dependent properties.
Example: *"In all the reachable states of the system, property A never holds"*

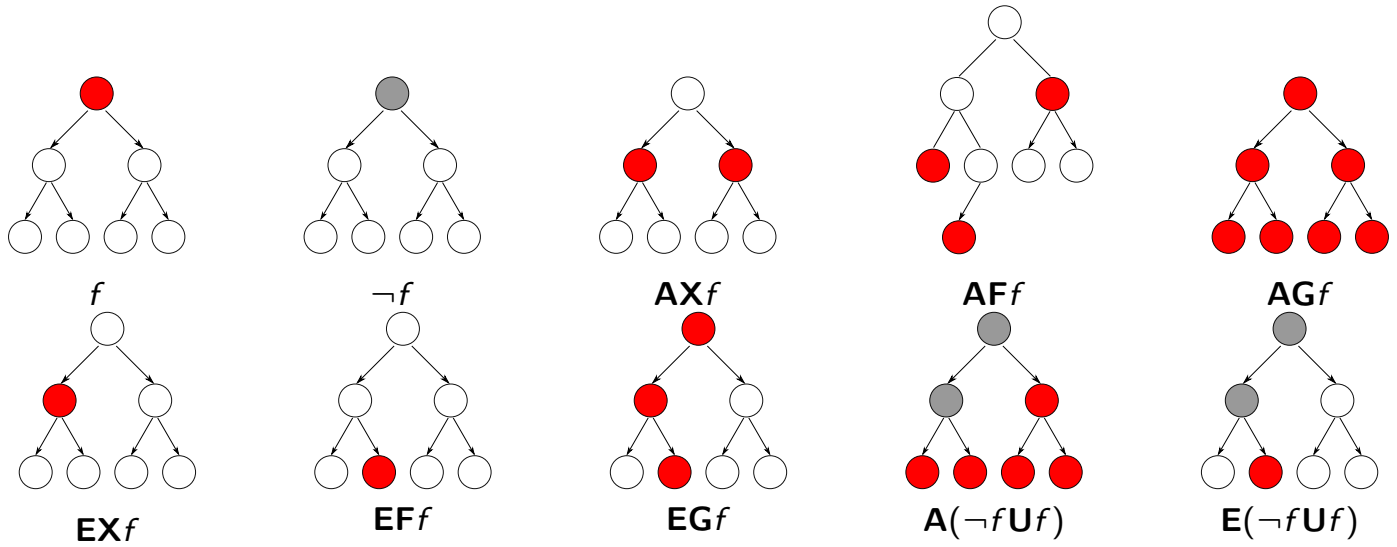
(a) LTL

(b) CTL



Quantum Model Checking (3)

Temporal Operators



Invariant and Eventually

A (i.e., for all computation paths) and **E** (i.e., eventually, for some computation path).

Quantum Model Checking (4)

QCTL

Quantum Computation Tree Logic, it provides also the operators:

- $\mathbb{Q}_{\sim\epsilon}[g]$;
- $Q =?[g]$;
- $qeval((Q =?)[g], \rho)$;
- $qprob((Q =?)[g], \rho) = tr(qeval((Q =?)[g], \rho)))$.

QCTL

Quantum Computation Tree Logic

A QCTL formula is a formula over the following grammar:

$$\Phi ::= a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathbb{Q}_{\sim\varepsilon}[\Phi] \qquad \textit{state} \text{ formula}$$

$$\phi ::= X\Phi \mid \Phi U^{\leq k}\Phi \mid \Phi U\Phi \qquad \textit{path} \text{ formula}$$

where $a \in AP$, $\sim \in \{\lesssim, \gtrsim, \eqsim\}$, $\mathcal{E} \in \mathcal{S}^{\mathcal{I}}(\mathcal{H})$, $k \in \mathbb{N}$.

Example

$Q >= 1\ [F(s = 5)]$

What we did: from Circuits to QMCs

Quip-E

We isolated and extended a Quipper fragment that we called *Quip-E* which allows the definition of both standard and *tail recursive* quantum programs.

Entangλe

We defined a mapping from *Quip-E* programs to QMCs. We start by considering a quantum program generated by *Quip-E* and we define a bisimilar QMC.

Formal definition of *Quip-E* program

Definition

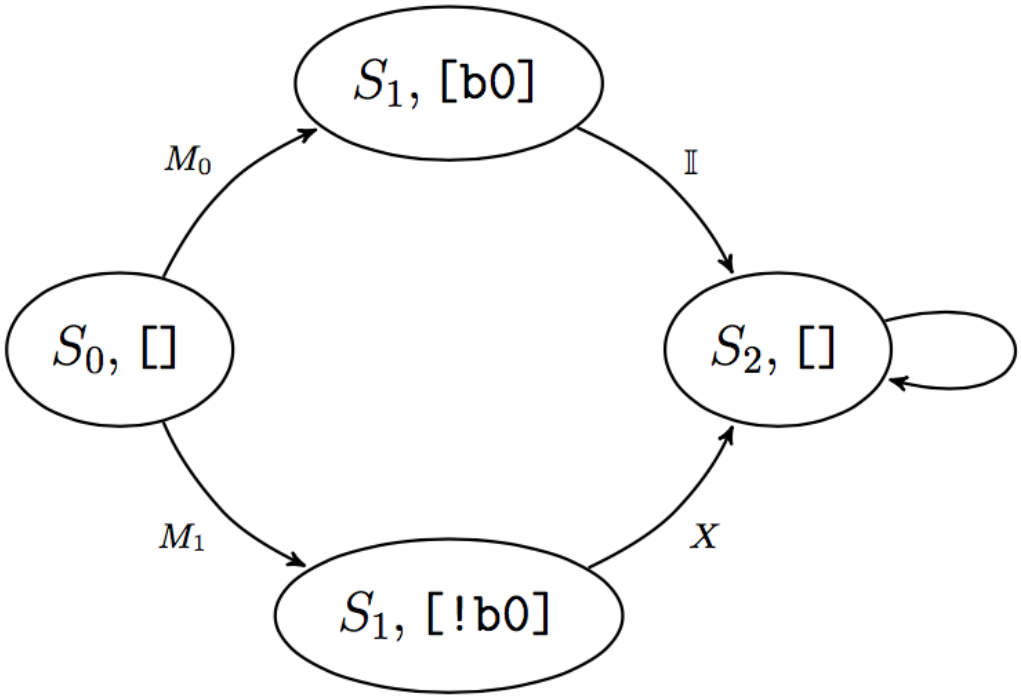
A *Quip-E* program is a circuit in which the result of a measurement is evaluated and could result in a loop.

Body of *Quip-E* program

- **reset**: initializes the qubits to $|0\rangle$;
- **unitary**: unitary operator applied to a list of qubits;
- **measure**: application of measurement operators to a list of qubits resulting in a list of bits;
- **dynamic lift**: A bit is lifted to a boolean through the dynamic lift Quip- per operator;
- **if-then-else**: evaluation of a Boolean expression;
- **exit On**: loop instruction.

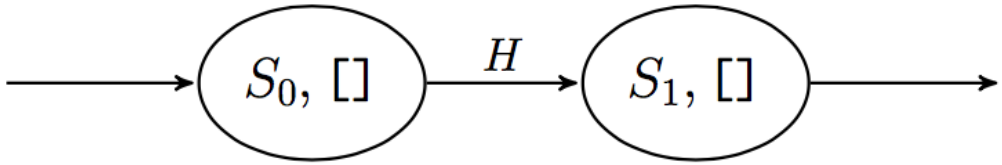
From *Quip-E* to QMC - intuitively

```
resetCirc :: Qubit -> Circ Qubit
resetCirc q = do
  reset_at q
```



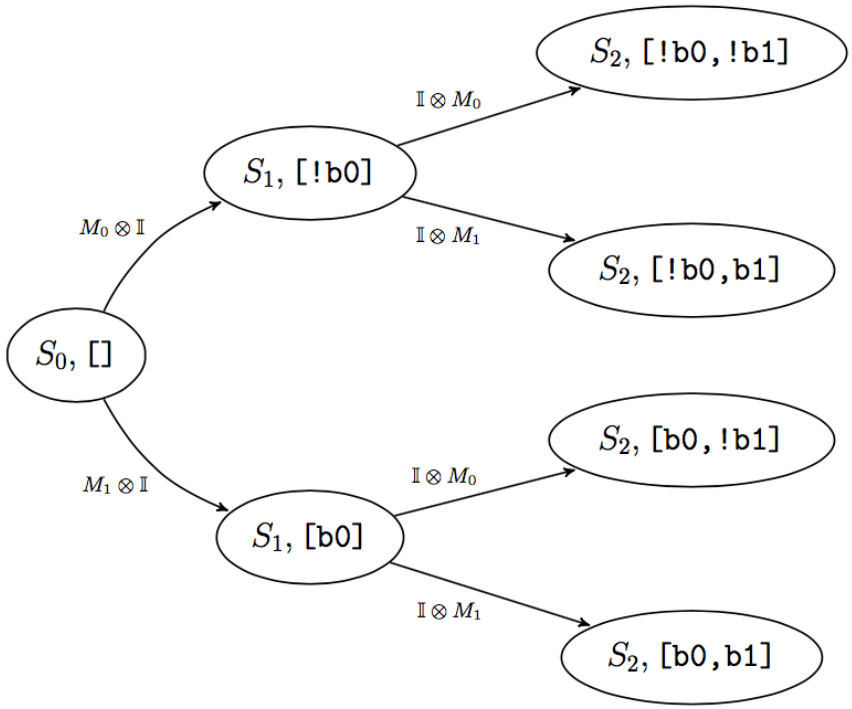
From *Quip-E* to QMC - intuitively

```
unitCirc :: Qubit -> Circ Qubit
unitCirc q = do
  hadamard_at q
```



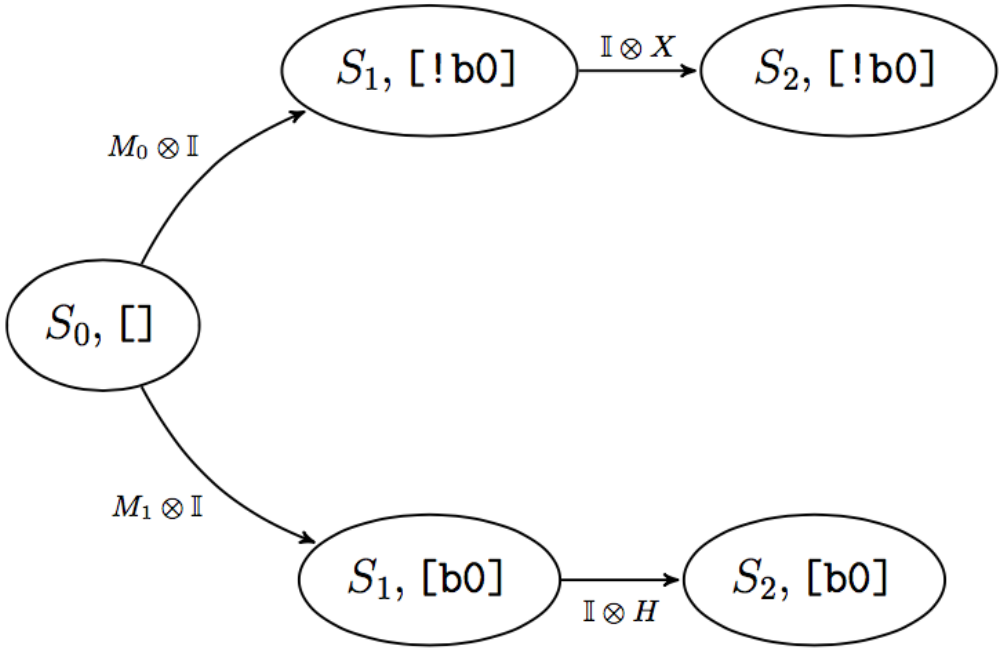
From *Quip-E* to QMC - intuitively

```
measureCirc :: (Qubit, Qubit) -> Circ ()
measureCirc (q1, q2) = do
  c1 <- measure q1
  c2 <- measure q2
```



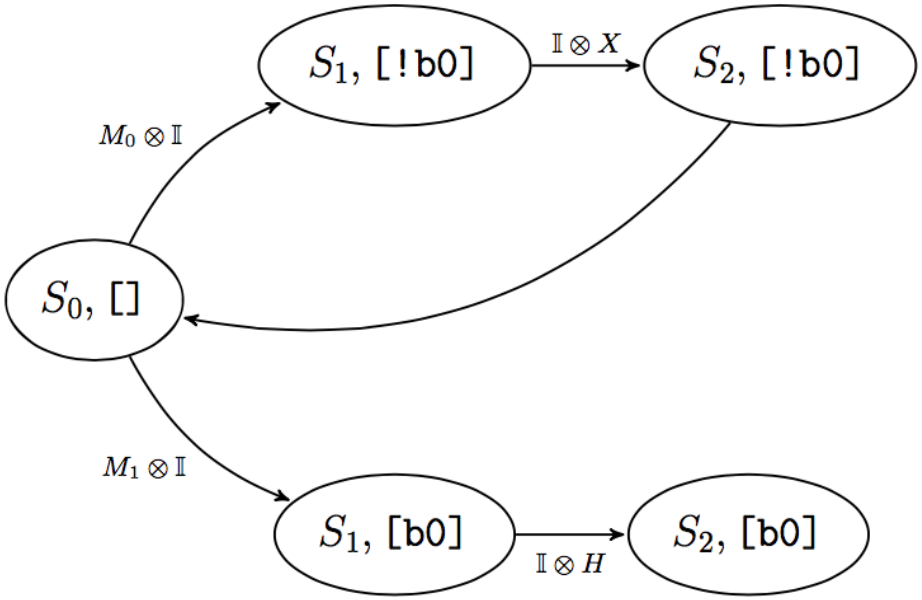
From *Quip-E* to QMC - intuitively

```
iteCirc :: (Qubit, Qubit) -> Circ ()
iteCirc (q1, q2) = do
  c1 <- measure q1
  b0 <- dynamic_lift c1
  if b0
    then do hadamard_at q2
    else do gate_X_at q2
```



From *Quip-E* to QMC - intuitively

```
loopCirc :: (Qubit, Qubit) -> Circ RecAction
loopCirc (q1, q2) = do
  c1 <- measure q1
  b0 <- dynamic_lift c1
  if b0
    then do hadamard_at q2
    else do gate_X_at q2
```



Operational Semantics of Quip–E (1)

$$\frac{}{(\texttt{reset\_at}\ \ \texttt{q}_k, L) \xrightarrow{\mathcal{M}_0^k} (\texttt{\_\_\_}, L)} \qquad \frac{}{(\texttt{reset\_at}\ \ \texttt{q}, L) \xrightarrow{\mathcal{M}_1^k} ((\texttt{X\_at}\ \ \texttt{q}_k, L)}$$

$$\frac{}{(\texttt{U\_at}\ \ [\texttt{q}_{i_1}, \ldots, \texttt{q}_{i_j}], L) \xrightarrow{\mathcal{U}_{i_1,\ldots,i_j}} (\texttt{\_\_\_}, L)}$$

$$\frac{}{(\texttt{m} \leftarrow \texttt{measure}\ \ \texttt{q}_k, L) \xrightarrow{\mathcal{M}_i^k} (\texttt{\_\_\_}, L[L(\texttt{m}) = i]\})} \quad \text{for } i \in \{0, 1\}$$

$$\frac{}{(\texttt{bool} \texttt{ <- } \texttt{dynamic\_lift}\ \ \texttt{m}, L) \xrightarrow{\mathcal{I}} (\texttt{\_\_\_}, L[L(\texttt{bool}) = L(\texttt{m})])}$$

Operational Semantics of Quip–E (2)

$$\frac{L(\texttt{bool}) = i}{(\texttt{if (bool) Body\_C}_1 \texttt{ else Body\_C}_0, L) \xrightarrow{\mathcal{I}} (\texttt{Body\_C}_i, L)} \quad \text{for } i \in \{0, 1\}$$

$$\frac{(\texttt{Body\_C}_1, L) \xrightarrow{\mathcal{S}} (\texttt{Body\_C}_1\texttt{'}, L')}{(\texttt{Body\_C}_1 \ \texttt{Body\_C}_2, L) \xrightarrow{\mathcal{S}} (\texttt{Body\_C}_1\texttt{'} \ \texttt{Body\_C}_2, L')}$$

$$\frac{(\texttt{Body\_C}_1, L) \xrightarrow{\mathcal{S}} (\_\_\_\_, L')}{(\texttt{Body\_C}_1 \ \texttt{Body\_C}_2, L) \xrightarrow{\mathcal{S}} (\texttt{Body\_C}_2, L')}$$

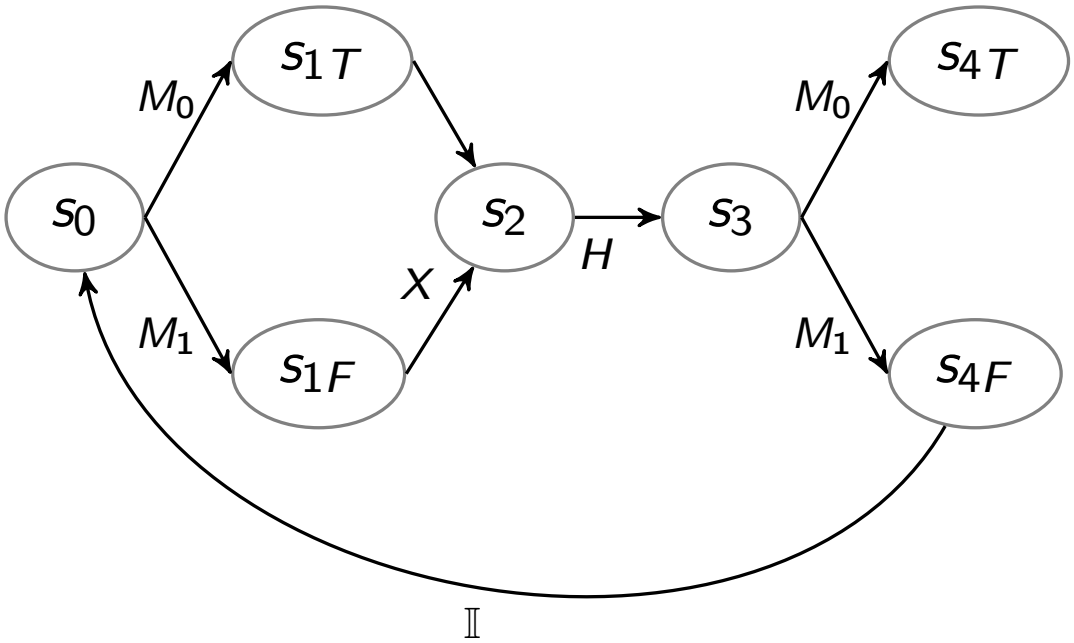
$$\frac{}{(\_\_\_\_, L) \xrightarrow{\mathcal{I}} (\_\_\_\_, L)}$$

Implementation

We implemented `Entangλe` using the `Transformer` module of Quipper. The input quantum program is a *Quip-E* function and the output QMC is a QPMC model.

1. The gates in the quantum circuit are grouped together with their associated qubits, preserving the execution order;
2. we compute the matrix representation of the quantum gates, taking into account also the conditional branches and the initialization operators;
3. the last step is the conversion of the list of transitions into QPMC code.

```
testInit :: (Qubit) -> Circ RecAction
testInit (q) = do
  reset_at q
  hadamard_at q
  ma <- measure q
  bool <- dynamic_lift ma
  exitOn bool
```




```
qmc

const matrix A1_T = M0;
const matrix A1_F = M1;
const matrix A2 = PauliX;
const matrix A3 = Hadamard;
const matrix A4_F = M0;
const matrix A4_T = M1;

module testInit
  s: [0..4] init 0;
  b0: bool init false;

  [] (s = 0) -> <<A1_T>> : (s' = 1) &
       (b0' = true) + <<A1_F>> : (s'
       = 1) & (b0' = false);
  [] (s = 1) & b0 -> (s' = 2);
  [] (s = 1) & !b0 -> <<A2>> : (s' =
       2);
  [] (s = 2) -> <<A3>> : (s' = 3);
  [] (s = 3) -> <<A4_F>> : (s' = 4) &
       (b0' = false) + <<A4_T>> : (s'
        = 4) & (b0' = true);
  [] (s = 4) & !b0 -> (s' = 0);
  [] (s = 4) & b0 -> true;
endmodule
```

Conclusion and Questions

TO–DO

- optimization of Entangλe to verify more complex quantum programs;
- optimization from the model checking point of view, involving the automatic verification of more complex properties, i.e., entanglement and other quantum effects;
- translation and verification of more complex, real-world quantum protocols;
- simulation (and translation) of quantum dynamics;
- spatial properties verification.