A Brief Introduction to Probabilistic and Quantum Programming

Part II

Ugo Dal Lago

Universidade do Minho, Friday May 5, 2017

Section 1

Probabilistic Programming Languages

Flipping a Coin

- Making any programming language probabilistic is relatively **easy**, at least from a purely **linguistic** point of view.
- The naïve solution is to endow your favourite language with a primitive that, when executed, "flips a fair coin" returning 0 or 1 with equal probability.
  - In **imperative** programming langauges, this can take the form of a keyword `rand`, to be used in expressions;
  - In **functional** programming languages, one could also use a form of binary, probabilistic sum, call it $\oplus$, as follows:

    ```
    letrec f x = x (+) (f (x+1))
    ```
- How do we get the necessary randomness, when executing programs?
  - By a source of *true* randomness, like physical randomness.
  - By *pseudorandomness* (we will come to that later).

Sampling

- If incepted into a universal programming language, binary uniform choice is enough to encode sampling from any **computable** distribution.
- As an example, if `f` is defined as follows

  ```
  letrec f x = x (+) (f (x+1))
  ```

  then `f(0)` produces the exponential distribution

$$\{0^{\frac{1}{2}}, 1^{\frac{1}{4}}, 2^{\frac{1}{8}}, \ldots\}.$$

- A **real number** $x \in \mathbb{R}$ is **computable** if there is an algorithm $\mathcal{A}_x$ outputting, on input $n \in \mathbb{N}$, a rational number $q_n \in \mathbb{Q}$ such that $|x - q_n| < \frac{1}{2^n}$.
- A **computable distribution** is one such that there is an algorithm $\mathcal{B}$ that, on input n, outputs the code of $\mathcal{A}_{p^n}$, where p^n is the probability the distribution assigns to n.

Theorem

PTMs are universal for computable distributions.

True Randomness vs. Pseudorandomness

- Having access to a source of true randomness is definitely not trivial.
- One could make use, as an example, of:
  - Keyboard and mouse actions;
  - External sources of randomness, like sound or movements;
  - TRNGs (True Random Number Generators).
- A **pseudorandom generator** is a deterministic algorithm $\mathcal{G}$ from strings in Σ^* to Σ^* such that:
  - $|\mathcal{G}(s)| > |s|$;
  - $\mathcal{G}(s)$ is somehow **indistinguishable** from a truly random string t *of the same length.*
- The point of pseudorandomness is **amplification**: a short truly random string is turned into a longer string which is not random, but looks so.

Programming vs. Modeling

- Probabilistic *models*, contrarily to probabilistic *languages*, are pervasive and very-well studied from decades.
  - Markov Chains
  - Markov Processes
  - Stochastic Processes
  - ...
- A probabilistic **program**, however, can indeed be seen as a way to concisely specify a model, for the purpose of doing, e.g., machine learning or inference.
  - A quite large research community is currently involved in this effort (for more details, see `http://probabilistic-programming.org`).
  - Roughly, you cannot only "flip a coin", but you can also incorporate **observations** about your dataset in your program.

Programming vs. Modeling

- Probabilistic *models*, contrarily to probabilistic *languages*, are pervasive and very-well studied from decades.
  - Markov Chains
  - Markov Processes
  - Stochastic Processes
  - ...
- A probabilistic **program**, however, can indeed be seen as a way to concisely specify a model, for the purpose of doing, e.g., machine learning or inference.
  - A quite large research community is currently involved in this effort (for more details, see `http://probabilistic-programming.org`).
  - Roughly, you cannot only "flip a coin", but you can also incorporate **observations** about your dataset in your program.

A Nice Example: FUN

```
let heads1 = random (Bernoulli(0.5)) in
let heads2 = random (Bernoulli(0.5)) in
let u = observe (heads1 || heads2) in
(heads1,heads2)
```

A Nice Example: FUN

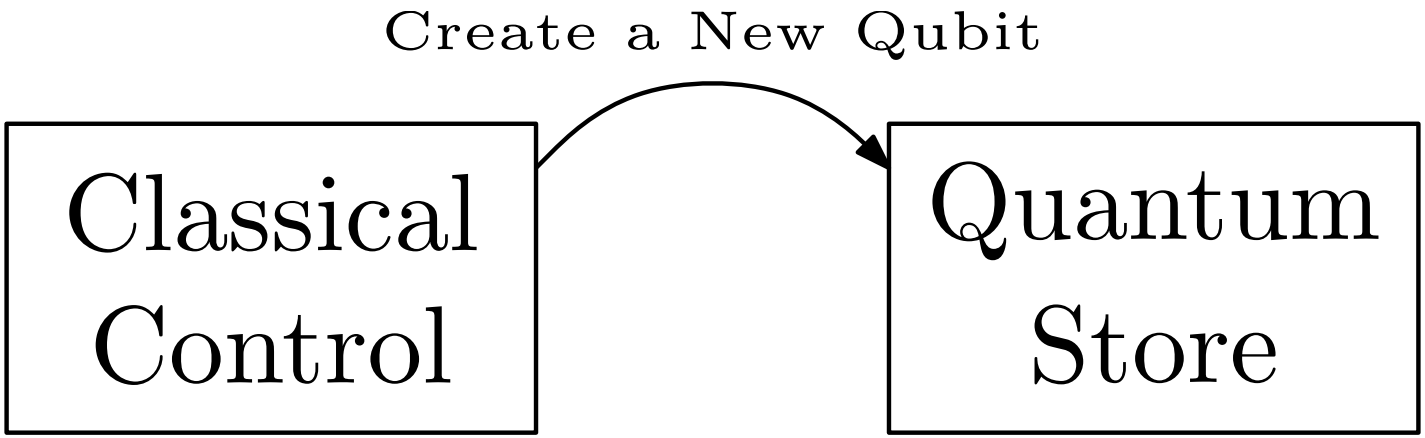
```
// prior distributions, the hypothesis
let skill() = random (Gaussian(10.0,20.0))
let Alice,Bob,Cyd = skill(),skill(),skill()
// observe the evidence
let performance player = random (Gaussian(player,1.0))
observe (performance Alice > performance Bob) //Alice beats Bob
observe (performance Bob > performance Cyd) //Bob beats Cyd
observe (performance Alice > performance Cyd) //Alice beats Cyd
// return the skills
Alice,Bob,Cyd
```

Section 2

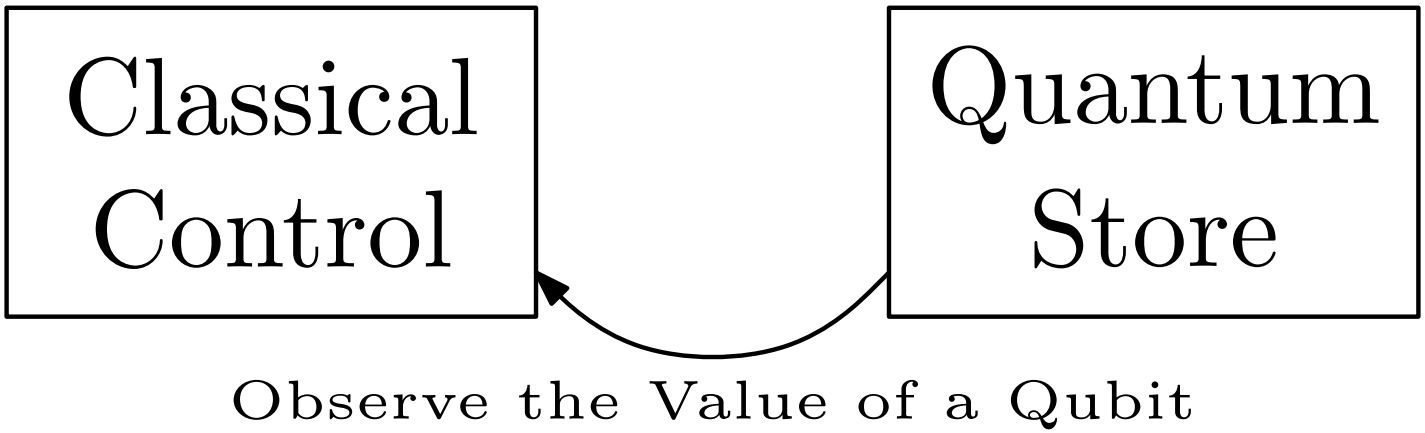
Quantum Programming Languages

Quantum Data and Classical Control

Quantum Data and Classical Control



Quantum Data and Classical Control



Quantum Data and Classical Control

Imperative Quantum Programming Languages

- QCL, which has been introduced by Ömer
- Example:

```
qufunct set(int n,qureg q)
  {
  int i;
  for i=0 to #q-1
    {
      if bit(n,i) {Not(q[i]);}
    }
  }
```

- Classical and quantum variables.
- The syntax is very reminiscent of the one of C.

Quantum Imperative Programming Languages

- qGCL, which has been introduced by Sanders and Zuliani.
  - It is based on Dijkstra's predicate transformers and guarded-command language, called GCL.
  - Features quantum, probabilistic, and nondeterministic evolution.
  - It can be seen as a generalization of pGCL, itself a *probabilistic* variation on GCL.
- Tafliovich and Hehner adapted predicative programming to quantum computation.
  - Predicative programming is not a proper programming language, but rather a methodology for specification and verification.

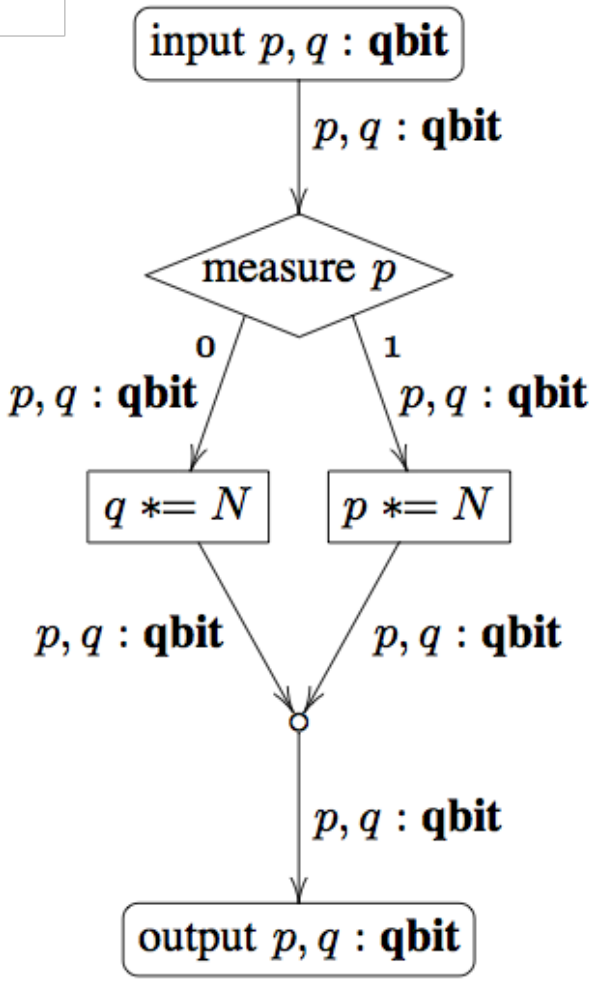
Quantum Functional Programming Languages

- QPL, introduced by Selinger.
  - A very simple, first-order, functional programming language.
  - The first one with a proper denotational semantics, given in terms of superoperators, but also handling divergence by way of domain theory.
  - A *superoperator* is a mathematical object by which we can describe the evolution of a quantum system **in presence of measurements**.
- Many papers investigated the possibility of embedding quantum programming into Haskell, arguably the most successful real-world functional programming language.
  - In a way or another, they are all based on the concept of a **monad**.

Quantum Functional Programming Languages

- QPL, introduced by Selinger.
  - A very simple, first-order, functional programming language.
  - The first one with a proper denotational semantics, given in terms of superoperators, but also handling divergence by way of domain theory.
  - A *superoperator* is a mathematical object by which we can describe the evolution of a quantum system **in presence of measurements**.
- Many papers investigated the possibility of embedding quantum programming into Haskell, arguably the most successful real-world functional programming language.
  - In a way or another, they are all based on the concept of a **monad**.

QPL: an Example



Quantum Functional Programming Languages

- In a first-order fragment of Haskell, one can also model a form of *quantum control*, i.e., programs whose internal **state** is in superposition.
  - This is Altenkirch and Grattage's QML.
- Operations are programmed at a very low level: unitary transforms become programs themselves, e.g.

$$
\begin{aligned}
&\mathit{had} : \mathbf{Q_2} \multimap \mathbf{Q_2} \\
&\mathit{had}\ x = \mathbf{if}^{\circ}\ x \\
&\qquad\qquad \mathbf{then}\ \{\mathrm{qfalse} \mid (-1)\ \mathrm{qtrue}\} \\
&\qquad\qquad \mathbf{else}\ \ \{\mathrm{qfalse} \mid \mathrm{qtrue}\}
\end{aligned}
$$

- Whenever you program by way of the *if* construct, you should be careful and check that the two branches are **orthogonal** in a certain sense.

Quantum Functional Programming Languages

- Most work on functional programming languages has focused on **λ-calculi**, which are *minimalist*, paradigmatic languages only including the essential features.
- Programs are seen as terms from a simple grammar

$$M, N ::= x \mid MN \mid \lambda x.M \mid \dots$$

- Computation is captured by way of **rewriting**
- Quantum features can be added in many different ways.
  - By adding **quantum variables**, which are meant to model the interaction with the quantum store.
  - By allowing terms to be in **superposition**, somehow diverging from the quantum-data-and-classical-control paradigm:

$$M ::= \dots \mid \sum_{i \in I} \alpha_i M_i$$

- The rest of this course will be almost entirely devoted to (probabilistic and) quantum λ-calculi.

Quantum Process Algebras

- Process algebras are calculi meant to model **concurrency** and **interaction** rather than mere computation.
- Terms of process algebras are usually of the following form;

$$P, Q ::= \mathbf{0} \mid a.P \mid \overline{a}.P \mid P||Q \mid \ldots$$

- Again, computation is modeled by a form of rewriting, e.g.,

$$a.P||\overline{a}.Q \to P||Q$$

- How could we incept quantum computation? Usually:
  - Each process has its own set of classical and quantum (local) variables.
  - Processes do not only synchronize, but can also **send** classical and quantum data along channels.
  - Unitary transformations and measurements are done locally.

Other Programming Paradigms

- **Concurrent Constraint Programming**
- **Measurement-Based Quantum Computation**
- **Hardware Description Languages**
- ...

Section 3

The λ-Calculus as a Functional Language

Minimal Syntax and Dynamics

- **Terms**:

$$M ::= x \mid MN \mid \lambda x.M.$$

- **Substitution**:

$$x\{x/M\} = M \qquad y\{x/M\} = y$$
$$NL\{x/M\} = (N\{x/M\})(L\{x/M\}) \qquad \lambda y.N\{x/M\} = \lambda y.(N\{x/M\})$$

- **CBN Operational Semantics**:

$$\frac{}{(\lambda x.M)N \to M\{x/N\}} \qquad \frac{M \to N}{ML \to NL}$$

- **Values**:

$$V ::= \lambda x.M$$

- **CBV Operational Semantics**:

$$\frac{}{(\lambda x.M)V \to M\{x/V\}} \qquad \frac{M \to N}{ML \to NL} \qquad \frac{M \to N}{VM \to VN}$$

Minimal Syntax and Dynamics

- **Terms**:

$$M ::= x \mid MN \mid \lambda x.M.$$

- **Substitution**:

$$x\{x/M\} = M \qquad y\{x/M\} = y$$
$$NL\{x/M\} = (N\{x/M\})(L\{x/M\}) \qquad \lambda y.N\{x/M\} = \lambda y.(N\{x/M\})$$

- **CBN Operational Semantics**:

$$\frac{}{(\lambda x.M)N \to M\{x/N\}} \qquad \frac{M \to N}{ML \to NL}$$

- **Values**:

$$V ::= \lambda x.M$$

- **CBV Operational Semantics**:

$$\frac{}{(\lambda x.M)V \to M\{x/V\}} \qquad \frac{M \to N}{ML \to NL} \qquad \frac{M \to N}{VM \to VN}$$

Minimal Syntax and Dynamics

- **Terms**:

$$M ::= x \mid MN \mid \lambda x.M.$$

- **Substitution**:

$$x\{x/M\} = M \qquad y\{x/M\} = y$$
$$NL\{x/M\} = (N\{x/M\})(L\{x/M\}) \qquad \lambda y.N\{x/M\} = \lambda y.(N\{x/M\})$$

- **CBN Operational Semantics**:

$$\frac{}{(\lambda x.M)N \to M\{x/N\}} \qquad \frac{M \to N}{ML \to NL}$$

- **Values**:

$$V ::= \lambda x.M$$

- **CBV Operational Semantics**:

$$\frac{}{(\lambda x.M)V \to M\{x/V\}} \qquad \frac{M \to N}{ML \to NL} \qquad \frac{M \to N}{VM \to VN}$$

Minimal Syntax and Dynamics

- **Terms**:

$$M ::= x \mid MN \mid \lambda x.M.$$

- **Substitution**:

$$x\{x/M\} = M \qquad y\{x/M\} = y$$
$$NL\{x/M\} = (N\{x/M\})(L\{x/M\}) \qquad \lambda y.N\{x/M\} = \lambda y.(N\{x/M\})$$

- **CBN Operational Semantics**:

$$\frac{}{(\lambda x.M)N \to M\{x/N\}} \qquad \frac{M \to N}{ML \to NL}$$

- **Values**:

$$V ::= \lambda x.M$$

- **CBV Operational Semantics**:

$$\frac{}{(\lambda x.M)V \to M\{x/V\}} \qquad \frac{M \to N}{ML \to NL} \qquad \frac{M \to N}{VM \to VN}$$

Minimal Syntax and Dynamics

- **Terms**:

$$M ::= x \mid MN \mid \lambda x.M.$$

- **Substitution**:

$$x\{x/M\} = M \qquad y\{x/M\} = y$$
$$NL\{x/M\} = (N\{x/M\})(L\{x/M\}) \qquad \lambda y.N\{x/M\} = \lambda y.(N\{x/M\})$$

- **CBN Operational Semantics**:

$$\frac{}{(\lambda x.M)N \to M\{x/N\}} \qquad \frac{M \to N}{ML \to NL}$$

- **Values**:

$$V ::= \lambda x.M$$

- **CBV Operational Semantics**:

$$\frac{}{(\lambda x.M)V \to M\{x/V\}} \qquad \frac{M \to N}{ML \to NL} \qquad \frac{M \to N}{VM \to VN}$$

Observations

- Not all redexes are reduced.
- For every M there is **at most** one N such that $M \to N$.
- It is easy to find a term M such that

$$M = N_0 \to N_1 \to N_2 \to \dots$$

- There is a problem with **substitutions**:

$$(\lambda x.\lambda y.x)(yz) \to \lambda y.yz.$$
$$(\lambda x.\lambda y.x)(yz) \to \lambda w.yz.$$

- The standard solution is to consider terms modulo so-called α**-equivalence**: renaming bound variables turns a term into one which is indistinguishable.

$$\lambda x.\lambda y.xxy \equiv \lambda z.\lambda y.zzy.$$

- α-equivalence is not necessary if we assume to work with **closed** terms.

Observations

- Not all redexes are reduced.
- For every M there is **at most** one N such that $M \to N$.
- It is easy to find a term M such that

$$M = N_0 \to N_1 \to N_2 \to \dots$$

- There is a problem with **substitutions**:

$$(\lambda x.\lambda y.x)(yz) \to \lambda y.yz.$$
$$(\lambda x.\lambda y.x)(yz) \to \lambda w.yz.$$

- The standard solution is to consider terms modulo so-called α**-equivalence**: renaming bound variables turns a term into one which is indistinguishable.

$$\lambda x.\lambda y.xxy \equiv \lambda z.\lambda y.zzy.$$

- α-equivalence is not necessary if we assume to work with **closed** terms.

Observations

- Not all redexes are reduced.
- For every M there is **at most** one N such that $M \to N$.
- It is easy to find a term M such that

$$M = N_0 \to N_1 \to N_2 \to \dots$$

- There is a problem with **substitutions**:

$$(\lambda x.\lambda y.x)(yz) \to \lambda y.yz.$$
$$(\lambda x.\lambda y.x)(yz) \to \lambda w.yz.$$

- The standard solution is to consider terms modulo so-called α**-equivalence**: renaming bound variables turns a term into one which is indistinguishable.

$$\lambda x.\lambda y.xxy \equiv \lambda z.\lambda y.zzy.$$

- α-equivalence is not necessary if we assume to work with **closed** terms.

Computing Simple Function on $\mathbb{N}$, in CBV

- **Natural Numbers**

$$\ulcorner 0 \urcorner = \lambda x.\lambda y.x;$$
$$\ulcorner n+1 \urcorner = \lambda x.\lambda y.y \ulcorner n \urcorner.$$

- **Finite Set** $A = \{a_1, \ldots, a_n\}$

$$\ulcorner a_i \urcorner = \lambda x_1 \cdots \lambda x_n.x_i.$$

- **Pairs**

$$\langle M, N \rangle = \lambda x.xMN$$

- **Fixed-Point Combinator**

$$\Theta = \lambda x.\lambda y.y(\lambda z.(xx)yz);$$
$$\mathsf{H} = \Theta\Theta.$$

Computing Simple Functions on $\mathbb{N}$, in CBV

- **Successor**

$$\mathsf{SUCC} = \lambda z.\lambda x.\lambda y.yz$$

Indeed:

$$\mathsf{SUCC}\ulcorner n\urcorner \to \lambda x.\lambda y.y\ulcorner n\urcorner$$

- **Projections**

$$\mathsf{PROJ}_i^k = \lambda x.x(\lambda y_1 \cdot \lambda y_k.y_i)$$

Indeed:

$$\begin{aligned}\mathsf{PROJ}_i^k\langle\ulcorner n_1\urcorner,\ldots,\ulcorner n_k\urcorner\rangle &\to \langle\ulcorner n_1\urcorner,\ldots,\ulcorner n_k\urcorner\rangle(\lambda y_1\cdots\lambda y_k.y_i)\\ &\to (\lambda y_1\cdots\lambda y_k.y_i)\ulcorner n_1\urcorner\cdots\ulcorner n_k\urcorner\\ &\to^* \ulcorner n_i\urcorner.\end{aligned}$$

Computing Simple Functions on $\mathbb{N}$, in CBV

- **Successor**

$$\mathsf{SUCC} = \lambda z.\lambda x.\lambda y.yz$$

Indeed:

$$\mathsf{SUCC}\ulcorner n\urcorner \to \lambda x.\lambda y.y\ulcorner n\urcorner$$

- **Projections**

$$\mathsf{PROJ}_i^k = \lambda x.x(\lambda y_1 \cdot \lambda y_k.y_i)$$

Indeed:

$$\begin{aligned}\mathsf{PROJ}_i^k\langle\ulcorner n_1\urcorner,\ldots,\ulcorner n_k\urcorner\rangle &\to \langle\ulcorner n_1\urcorner,\ldots,\ulcorner n_k\urcorner\rangle(\lambda y_1\cdots\lambda y_k.y_i)\\ &\to (\lambda y_1\cdots\lambda y_k.y_i)\ulcorner n_1\urcorner\cdots\ulcorner n_k\urcorner\\ &\to^* \ulcorner n_i\urcorner.\end{aligned}$$

Computing Simple Functions on $\mathbb{N}$, in CBV

- **Successor**

$$\mathsf{SUCC} = \lambda z.\lambda x.\lambda y.yz$$

Indeed:

$$\mathsf{SUCC}\ulcorner n\urcorner \to \lambda x.\lambda y.y\ulcorner n\urcorner$$

- **Projections**

$$\mathsf{PROJ}_i^k = \lambda x.x(\lambda y_1 \cdot \lambda y_k.y_i)$$

Indeed:

$$\begin{aligned}\mathsf{PROJ}_i^k\langle\ulcorner n_1\urcorner,\ldots,\ulcorner n_k\urcorner\rangle &\to \langle\ulcorner n_1\urcorner,\ldots,\ulcorner n_k\urcorner\rangle(\lambda y_1\cdots\lambda y_k.y_i)\\ &\to (\lambda y_1\cdots\lambda y_k.y_i)\ulcorner n_1\urcorner\cdots\ulcorner n_k\urcorner\\ &\to^* \ulcorner n_i\urcorner.\end{aligned}$$

Computing Simple Functions on $\mathbb{N}$, in CBV

- **Successor**

$$\mathsf{SUCC} = \lambda z.\lambda x.\lambda y.yz$$

Indeed:

$$\mathsf{SUCC}\ulcorner n\urcorner \to \lambda x.\lambda y.y\ulcorner n\urcorner$$

- **Projections**

$$\mathsf{PROJ}_i^k = \lambda x.x(\lambda y_1 \cdot \lambda y_k.y_i)$$

Indeed:

$$\begin{aligned}\mathsf{PROJ}_i^k\langle\ulcorner n_1\urcorner,\ldots,\ulcorner n_k\urcorner\rangle &\to \langle\ulcorner n_1\urcorner,\ldots,\ulcorner n_k\urcorner\rangle(\lambda y_1\cdots\lambda y_k.y_i)\\ &\to (\lambda y_1\cdots\lambda y_k.y_i)\ulcorner n_1\urcorner\cdots\ulcorner n_k\urcorner\\ &\to^* \ulcorner n_i\urcorner.\end{aligned}$$

A Simple Form of Recursion

- Suppose you have a term M_f which computes the function $f : \mathbb{N} \to \mathbb{N}$, and you want to iterate over it, i.e. you want to compute $g : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that

$$g(0, n) = n;$$
$$g(m + 1, n) = f(g(m, n)).$$

- We need a form of recursion in the language. Observe that:

$$\mathsf{H}V = (\Theta\Theta)V \to (\lambda y.y(\lambda z.\mathsf{H}yz))V \to V(\lambda z.\mathsf{H}Vz)$$

- The function g can be computed by way of

$$M_g = \mathsf{H}(\lambda x.\lambda y.y(\lambda z.\lambda w.zw(\lambda q.M_f(x\langle q, w\rangle))))$$

A Simple Form of Recursion

- Suppose you have a term M_f which computes the function $f : \mathbb{N} \to \mathbb{N}$, and you want to iterate over it, i.e. you want to compute $g : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that

$$g(0, n) = n;$$
$$g(m + 1, n) = f(g(m, n)).$$

- We need a form of recursion in the language. Observe that:

$$\mathsf{H}V = (\Theta\Theta)V \to (\lambda y.y(\lambda z.\mathsf{H}yz))V \to V(\lambda z.\mathsf{H}Vz)$$

- The function g can be computed by way of

$$M_g = \mathsf{H}(\lambda x.\lambda y.y(\lambda z.\lambda w.zw(\lambda q.M_f(x\langle q, w\rangle))))$$

A Simple Form of Recursion

- Suppose you have a term M_f which computes the function $f : \mathbb{N} \to \mathbb{N}$, and you want to iterate over it, i.e. you want to compute $g : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that

$$g(0, n) = n;$$
$$g(m + 1, n) = f(g(m, n)).$$

- We need a form of recursion in the language. Observe that:

$$\mathsf{H}V = (\Theta\Theta)V \to (\lambda y.y(\lambda z.\mathsf{H}yz))V \to V(\lambda z.\mathsf{H}Vz)$$

- The function g can be computed by way of

$$M_g = \mathsf{H}(\lambda x.\lambda y.y(\lambda z.\lambda w.zw(\lambda q.M_f(x\langle q, w\rangle))))$$

Universality

- One could go on, and show that the following schemes can all be encoded:
  - **Composition**;
  - **Primitive Recursion**;
  - **Minimization**.
- This implies that all partial recursive functions can be computed in the λ-calculus.

Theorem

The λ-calculus is Turing-complete, both when CBV and CBN are considered.

Why is This a Faithful Model of Functional Computation?

- Data, conditionals, basic functions, and recursion can all be encoded into the λ-calculus. We can then give programs "informally", in our favourite functional language.
- Take, as an example, the usual recursive program computing the factorial of a natural number

```
let rec fact x = if (x==0) then 1 else x*(fact x-1)
```

- This can indeed be turned into a λ-term (where M_* and M_{-1} are terms for multiplication and the predecessor, respectively):

```
let rec fact x = if (x==0) then 1 else x*(fact x-1)
```

$\mathsf{H}(\lambda \texttt{fact}.\lambda \texttt{x}.\texttt{if (x==0) then 1 else x*(fact x-1)})$

$\mathsf{H}(\lambda \texttt{fact}.\lambda \texttt{x}.\texttt{x}\ulcorner 1 \urcorner (\texttt{x*(fact x-1)}))$

$\mathsf{H}(\lambda \texttt{fact}.\lambda \texttt{x}.\texttt{x}\ulcorner 1 \urcorner (M_* \texttt{ x (fact x-1)}))$

$\mathsf{H}(\lambda \texttt{fact}.\lambda \texttt{x}.\texttt{x}\ulcorner 1 \urcorner (M_* \texttt{ x (fact } (M_{-1} \texttt{x}))))$

Why is This a Faithful Model of Functional Computation?

- Data, conditionals, basic functions, and recursion can all be encoded into the λ-calculus. We can then give programs "informally", in our favourite functional language.
- Take, as an example, the usual recursive program computing the factorial of a natural number

```
let rec fact x = if (x==0) then 1 else x*(fact x-1)
```

- This can indeed be turned into a λ-term (where M_* and M_{-1} are terms for multiplication and the predecessor, respectively):

$$\texttt{let rec fact x = if (x==0) then 1 else x*(fact x-1)}$$

$$\mathsf{H}(\lambda\texttt{fact}.\lambda\texttt{x.if (x==0) then 1 else x*(fact x-1)})$$

$$\mathsf{H}(\lambda\texttt{fact}.\lambda\texttt{x.x}\ulcorner\texttt{1}\urcorner\,(\texttt{x*(fact x-1)}))$$

$$\mathsf{H}(\lambda\texttt{fact}.\lambda\texttt{x.x}\ulcorner\texttt{1}\urcorner\,(M_*\ \texttt{x}\ (\texttt{fact x-1})))$$

$$\mathsf{H}(\lambda\texttt{fact}.\lambda\texttt{x.x}\ulcorner\texttt{1}\urcorner\,(M_*\ \texttt{x}\ (\texttt{fact}\ (M_{-1}\texttt{x}))))$$

Why is This a Faithful Model of Functional Computation?

- Data, conditionals, basic functions, and recursion can all be encoded into the λ-calculus. We can then give programs "informally", in our favourite functional language.
- Take, as an example, the usual recursive program computing the factorial of a natural number

```
let rec fact x = if (x==0) then 1 else x*(fact x-1)
```

- This can indeed be turned into a λ-term (where M_* and M_{-1} are terms for multiplication and the predecessor, respectively):

$$\texttt{let rec fact x = if (x==0) then 1 else x*(fact x-1)}$$

$$\mathsf{H}(\lambda\texttt{fact}.\lambda\texttt{x.if (x==0) then 1 else x*(fact x-1)})$$

$$\mathsf{H}(\lambda\texttt{fact}.\lambda\texttt{x.x}\ulcorner 1\urcorner\,(\texttt{x*(fact x-1)}))$$

$$\mathsf{H}(\lambda\texttt{fact}.\lambda\texttt{x.x}\ulcorner 1\urcorner\,(M_*\ \texttt{x}\ (\texttt{fact x-1})))$$

$$\mathsf{H}(\lambda\texttt{fact}.\lambda\texttt{x.x}\ulcorner 1\urcorner\,(M_*\ \texttt{x}\ (\texttt{fact}\ (M_{-1}\texttt{x}))))$$

Why is This a Faithful Model of Functional Computation?

- Data, conditionals, basic functions, and recursion can all be encoded into the λ-calculus. We can then give programs "informally", in our favourite functional language.
- Take, as an example, the usual recursive program computing the factorial of a natural number

```
let rec fact x = if (x==0) then 1 else x*(fact x-1)
```

- This can indeed be turned into a λ-term (where M_* and M_{-1} are terms for multiplication and the predecessor, respectively):

$$\texttt{let rec fact x = if (x==0) then 1 else x*(fact x-1)}$$

$$\mathsf{H}(\lambda\texttt{fact}.\lambda\texttt{x.if (x==0) then 1 else x*(fact x-1)})$$

$$\mathsf{H}(\lambda\texttt{fact}.\lambda\texttt{x.x}\ulcorner 1\urcorner\,(\texttt{x*(fact x-1)}))$$

$$\mathsf{H}(\lambda\texttt{fact}.\lambda\texttt{x.x}\ulcorner 1\urcorner\,(M_*\ \texttt{x (fact x-1)}))$$

$$\mathsf{H}(\lambda\texttt{fact}.\lambda\texttt{x.x}\ulcorner 1\urcorner\,(M_*\ \texttt{x (fact}\ (M_{-1}\texttt{x}))))$$

Why is This a Faithful Model of Functional Computation?

- Data, conditionals, basic functions, and recursion can all be encoded into the λ-calculus. We can then give programs "informally", in our favourite functional language.
- Take, as an example, the usual recursive program computing the factorial of a natural number

```
let rec fact x = if (x==0) then 1 else x*(fact x-1)
```

- This can indeed be turned into a λ-term (where M_* and M_{-1} are terms for multiplication and the predecessor, respectively):

$$\texttt{let rec fact x = if (x==0) then 1 else x*(fact x-1)}$$

$$\mathsf{H}(\lambda\texttt{fact}.\lambda\texttt{x.if (x==0) then 1 else x*(fact x-1)})$$

$$\mathsf{H}(\lambda\texttt{fact}.\lambda\texttt{x.x}\ulcorner 1\urcorner\,(\texttt{x*(fact x-1)}))$$

$$\mathsf{H}(\lambda\texttt{fact}.\lambda\texttt{x.x}\ulcorner 1\urcorner\,(M_*\ \texttt{x (fact x-1)}))$$

$$\mathsf{H}(\lambda\texttt{fact}.\lambda\texttt{x.x}\ulcorner 1\urcorner\,(M_*\ \texttt{x (fact}\ (M_{-1}\texttt{x}))))$$

More About Semantics

- The semantics we have adopted so far is said to be **small-step**, since it derives assertions in the form $M \to N$, each corresponding to one atomic computation step.
- There is an equivalent way of formulating the same concept, **big-step semantics**, in which we "jump directly to the result":
  - **CBN**

$$\frac{}{V \Downarrow V} \qquad \frac{M \Downarrow \lambda x.N \qquad N\{x/L\} \Downarrow V}{ML \Downarrow V}$$

  - **CBV**

$$\frac{}{V \Downarrow V} \qquad \frac{M \Downarrow \lambda x.N \qquad L \Downarrow V \qquad N\{x/V\} \Downarrow W}{ML \Downarrow W}$$

Theorem

In both CBN and CBV, $M \to^* V$ if and only if $M \Downarrow V$.

- If $M \Downarrow V$ for *some* V, then we write $M \Downarrow$, otherwise $M \Uparrow$

More About Semantics

- The semantics we have adopted so far is said to be **small-step**, since it derives assertions in the form $M \to N$, each corresponding to one atomic computation step.
- There is an equivalent way of formulating the same concept, **big-step semantics**, in which we "jump directly to the result":
  - **CBN**

$$\frac{}{V \Downarrow V} \qquad \frac{M \Downarrow \lambda x.N \qquad N\{x/L\} \Downarrow V}{ML \Downarrow V}$$

  - **CBV**

$$\frac{}{V \Downarrow V} \qquad \frac{M \Downarrow \lambda x.N \qquad L \Downarrow V \qquad N\{x/V\} \Downarrow W}{ML \Downarrow W}$$

Theorem

In both CBN and CBV, $M \to^* V$ if and only if $M \Downarrow V$.

- If $M \Downarrow V$ for *some* V, then we write $M \Downarrow$, otherwise $M \Uparrow$

More About Semantics

- The semantics we have adopted so far is said to be **small-step**, since it derives assertions in the form $M \to N$, each corresponding to one atomic computation step.
- There is an equivalent way of formulating the same concept, **big-step semantics**, in which we "jump directly to the result":
  - **CBN**

$$\frac{}{V \Downarrow V} \qquad \frac{M \Downarrow \lambda x.N \qquad N\{x/L\} \Downarrow V}{ML \Downarrow V}$$

  - **CBV**

$$\frac{}{V \Downarrow V} \qquad \frac{M \Downarrow \lambda x.N \qquad L \Downarrow V \qquad N\{x/V\} \Downarrow W}{ML \Downarrow W}$$

Theorem

In both CBN and CBV, $M \to^* V$ if and only if $M \Downarrow V$.

- If $M \Downarrow V$ for *some* V, then we write $M \Downarrow$, otherwise $M \Uparrow$

Section 4

Probabilistic λ-Calculi

Probabilistic λ-Calculus: Syntax and Operational Semantics

- **Terms**: $M, N \ ::= \ x \ \mid \ \lambda x.M \ \mid \ MN \ \mid \ M \oplus N$;
- **Values**: $V \ ::= \ \lambda x.M$;
- How should we define the semantics of a probabilistic term M?
  - Informally, $M \oplus N$ rewrites to M or to N with probability $\frac{1}{2}$: this is just like flippling a coin and choosing one of the two terms according to the result.
  - A **value distribution** is a function $\mathscr{D}$ from the set $\mathcal{V}$ of values to $\mathbb{R}$ such
$$\sum_{V \in \mathcal{V}} \mathscr{D}(V) \leq 1.$$
  - The fact the sum can be **strictly smaller** than 1 is a way to reflect (the probability of) divergence.
  - The empty value distribution is denoted as $\emptyset$.
  - Useful notation for finite distributions: $\{V_1^{p_1}, \ldots, V_n^{p_n}\}$.
  - Value distributions can be ordered, pointwise.
  - $\mathsf{S}(\mathscr{D})$ is the **support** of $\mathscr{D}$, i.e. the set of values to which $\mathscr{D}$ assigns nonnull probability.

Probabilistic λ-Calculus: Syntax and Operational Semantics

- **Terms**: $M, N \ ::= \ x \ \mid \ \lambda x.M \ \mid \ MN \ \mid \ M \oplus N$;
- **Values**: $V \ ::= \ \lambda x.M$;
- How should we define the semantics of a probabilistic term M?
  - Informally, $M \oplus N$ rewrites to M or to N with probability $\frac{1}{2}$: this is just like flippling a coin and choosing one of the two terms according to the result.
  - A **value distribution** is a function $\mathscr{D}$ from the set $\mathcal{V}$ of values to $\mathbb{R}$ such
$$\sum_{V \in \mathcal{V}} \mathscr{D}(V) \leq 1.$$
  - The fact the sum can be **strictly smaller** than 1 is a way to reflect (the probability of) divergence.
  - The empty value distribution is denoted as $\emptyset$.
  - Useful notation for finite distributions: $\{V_1^{p_1}, \ldots, V_n^{p_n}\}$.
  - Value distributions can be ordered, pointwise.
  - $\mathsf{S}(\mathscr{D})$ is the **support** of $\mathscr{D}$, i.e. the set of values to which $\mathscr{D}$ assigns nonnull probability.

Probabilistic λ-Calculus: Syntax and Operational Semantics

- **Terms**: $M, N \ ::= \ x \ \mid \ \lambda x.M \ \mid \ MN \ \mid \ M \oplus N$;
- **Values**: $V \ ::= \ \lambda x.M$;
- How should we define the semantics of a probabilistic term M?
  - Informally, $M \oplus N$ rewrites to M or to N with probability $\frac{1}{2}$: this is just like flippling a coin and choosing one of the two terms according to the result.
  - A **value distribution** is a function $\mathscr{D}$ from the set $\mathcal{V}$ of values to $\mathbb{R}$ such
$$\sum_{V \in \mathcal{V}} \mathscr{D}(V) \leq 1.$$
  - The fact the sum can be **strictly smaller** than 1 is a way to reflect (the probability of) divergence.
  - The empty value distribution is denoted as $\emptyset$.
  - Useful notation for finite distributions: $\{V_1^{p_1}, \ldots, V_n^{p_n}\}$.
  - Value distributions can be ordered, pointwise.
  - $\mathsf{S}(\mathscr{D})$ is the **support** of $\mathscr{D}$, i.e. the set of values to which $\mathscr{D}$ assigns nonnull probability.

Probabilistic λ-Calculus: Syntax and Operational Semantics

- **Approximation (Big-Step) Semantics, CBN**:

$$\frac{}{M \Downarrow \emptyset} \qquad \frac{}{V \Downarrow \{V^1\}} \qquad \frac{M \Downarrow \mathscr{D} \qquad N \Downarrow \mathscr{E}}{M \oplus N \Downarrow \frac{1}{2}\mathscr{D} + \frac{1}{2}\mathscr{E}}$$

$$\frac{M \Downarrow \mathscr{D} \qquad \{P\{x/N\} \Downarrow \mathscr{E}_{\lambda x.P}\}_{\lambda x.P \in \mathsf{S}(\mathscr{D})}}{MN \Downarrow \sum_{V \in \mathsf{S}(\mathscr{F})} \mathscr{D}(V)\mathscr{E}_{\lambda x.P}}$$

- **Example**: consider $M = I \oplus ((II) \oplus \Omega)$, where $I = \lambda x.x$ and $\Omega = (\lambda x.xx)(\lambda x.xx)$.

$$M \Downarrow \emptyset \qquad M \Downarrow \{I^{\frac{1}{2}}\} \qquad M \Downarrow \{I^{\frac{3}{4}}\}$$

- **Semantics**: $[\![M]\!] = \sup_{M \Downarrow \mathscr{D}} \mathscr{D}$;
- Variations: **Small-Step** Semantics, **CBV**.

Probabilistic λ-Calculus: Syntax and Operational Semantics

- **Approximation (Big-Step) Semantics, CBN**:

$$\frac{}{M \Downarrow \emptyset} \qquad \frac{}{V \Downarrow \{V^1\}} \qquad \frac{M \Downarrow \mathscr{D} \qquad N \Downarrow \mathscr{E}}{M \oplus N \Downarrow \frac{1}{2}\mathscr{D} + \frac{1}{2}\mathscr{E}}$$

$$\frac{M \Downarrow \mathscr{D} \qquad \{P\{x/N\} \Downarrow \mathscr{E}_{\lambda x.P}\}_{\lambda x.P \in \mathsf{S}(\mathscr{D})}}{MN \Downarrow \sum_{V \in \mathsf{S}(\mathscr{F})} \mathscr{D}(V)\mathscr{E}_{\lambda x.P}}$$

- **Example**: consider $M = I \oplus ((II) \oplus \Omega)$, where $I = \lambda x.x$ and $\Omega = (\lambda x.xx)(\lambda x.xx)$.

$$M \Downarrow \emptyset \qquad M \Downarrow \{I^{\frac{1}{2}}\} \qquad M \Downarrow \{I^{\frac{3}{4}}\}$$

- **Semantics**: $[\![M]\!] = \sup_{M \Downarrow \mathscr{D}} \mathscr{D}$;
- Variations: **Small-Step** Semantics, **CBV**.

Probabilistic λ-Calculus: Syntax and Operational Semantics

- **Approximation (Big-Step) Semantics, CBN**:

$$\frac{}{M \Downarrow \emptyset} \qquad \frac{}{V \Downarrow \{V^1\}} \qquad \frac{M \Downarrow \mathscr{D} \qquad N \Downarrow \mathscr{E}}{M \oplus N \Downarrow \frac{1}{2}\mathscr{D} + \frac{1}{2}\mathscr{E}}$$

$$\frac{M \Downarrow \mathscr{D} \qquad \{P\{x/N\} \Downarrow \mathscr{E}_{\lambda x.P}\}_{\lambda x.P \in \mathsf{S}(\mathscr{D})}}{MN \Downarrow \sum_{V \in \mathsf{S}(\mathscr{F})} \mathscr{D}(V)\mathscr{E}_{\lambda x.P}}$$

- **Example**: consider $M = I \oplus ((II) \oplus \Omega)$, where $I = \lambda x.x$ and $\Omega = (\lambda x.xx)(\lambda x.xx)$.

$$M \Downarrow \emptyset \qquad M \Downarrow \{I^{\frac{1}{2}}\} \qquad M \Downarrow \{I^{\frac{3}{4}}\}$$

- **Semantics**: $[\![M]\!] = \sup_{M \Downarrow \mathscr{D}} \mathscr{D}$;
- Variations: **Small-Step** Semantics, **CBV**.

Probabilistic λ-Calculus: Syntax and Operational Semantics

- **Approximation (Big-Step) Semantics, CBN**:

$$\frac{}{M \Downarrow \emptyset} \qquad \frac{}{V \Downarrow \{V^1\}} \qquad \frac{M \Downarrow \mathscr{D} \qquad N \Downarrow \mathscr{E}}{M \oplus N \Downarrow \frac{1}{2}\mathscr{D} + \frac{1}{2}\mathscr{E}}$$

$$\frac{M \Downarrow \mathscr{D} \qquad \{P\{x/N\} \Downarrow \mathscr{E}_{\lambda x.P}\}_{\lambda x.P \in \mathsf{S}(\mathscr{D})}}{MN \Downarrow \sum_{V \in \mathsf{S}(\mathscr{F})} \mathscr{D}(V)\mathscr{E}_{\lambda x.P}}$$

- **Example**: consider $M = I \oplus ((II) \oplus \Omega)$, where $I = \lambda x.x$ and $\Omega = (\lambda x.xx)(\lambda x.xx)$.

$$M \Downarrow \emptyset \qquad M \Downarrow \{I^{\frac{1}{2}}\} \qquad M \Downarrow \{I^{\frac{3}{4}}\}$$

- **Semantics**: $[\![M]\!] = \sup_{M \Downarrow \mathscr{D}} \mathscr{D}$;
- Variations: **Small-Step** Semantics, **CBV**.

Probabilistic λ-Calculus: Syntax and Operational Semantics

- **Approximation (Big-Step) Semantics, CBN**:

$$\frac{}{M \Downarrow \emptyset} \qquad \frac{}{V \Downarrow \{V^1\}} \qquad \frac{M \Downarrow \mathscr{D} \qquad N \Downarrow \mathscr{E}}{M \oplus N \Downarrow \frac{1}{2}\mathscr{D} + \frac{1}{2}\mathscr{E}}$$

$$\frac{M \Downarrow \mathscr{D} \qquad \{P\{x/N\} \Downarrow \mathscr{E}_{\lambda x.P}\}_{\lambda x.P \in \mathsf{S}(\mathscr{D})}}{MN \Downarrow \sum_{V \in \mathsf{S}(\mathscr{F})} \mathscr{D}(V)\mathscr{E}_{\lambda x.P}}$$

- **Example**: consider $M = I \oplus ((II) \oplus \Omega)$, where $I = \lambda x.x$ and $\Omega = (\lambda x.xx)(\lambda x.xx)$.

$$M \Downarrow \emptyset \qquad M \Downarrow \{I^{\frac{1}{2}}\} \qquad M \Downarrow \{I^{\frac{3}{4}}\}$$

- **Semantics**: $[\![M]\!] = \sup_{M \Downarrow \mathscr{D}} \mathscr{D}$;
- Variations: **Small-Step** Semantics, **CBV**.

Probabilistic λ-Calculus: Syntax and Operational Semantics

- **Approximation (Big-Step) Semantics, CBN**:

$$\frac{}{M \Downarrow \emptyset} \qquad \frac{}{V \Downarrow \{V^1\}} \qquad \frac{M \Downarrow \mathscr{D} \qquad N \Downarrow \mathscr{E}}{M \oplus N \Downarrow \frac{1}{2}\mathscr{D} + \frac{1}{2}\mathscr{E}}$$

$$\frac{M \Downarrow \mathscr{D} \qquad \{P\{x/N\} \Downarrow \mathscr{E}_{\lambda x.P}\}_{\lambda x.P \in \mathsf{S}(\mathscr{D})}}{MN \Downarrow \sum_{V \in \mathsf{S}(\mathscr{F})} \mathscr{D}(V)\mathscr{E}_{\lambda x.P}}$$

- **Example**: consider $M = I \oplus ((II) \oplus \Omega)$, where $I = \lambda x.x$ and $\Omega = (\lambda x.xx)(\lambda x.xx)$.

$$M \Downarrow \emptyset \qquad M \Downarrow \{I^{\frac{1}{2}}\} \qquad M \Downarrow \{I^{\frac{3}{4}}\}$$

- **Semantics**: $[\![M]\!] = \sup_{M \Downarrow \mathscr{D}} \mathscr{D}$;
- Variations: **Small-Step** Semantics, **CBV**.

Probabilistic λ-Calculus: Syntax and Operational Semantics

- As an **example**, consider the following recursive program, and call it M:

```
let rec fancy x = x (+) (fancy x+1)
```

- One can easily check that:

$$M\ulcorner 0\urcorner \Downarrow \emptyset \qquad M\ulcorner 0\urcorner \Downarrow \{\ulcorner 0\urcorner^{\frac{1}{2}}\} \qquad M\ulcorner 0\urcorner \Downarrow \{\ulcorner 0\urcorner^{\frac{1}{2}}, \ulcorner 1\urcorner^{\frac{1}{4}}\} \qquad \dots$$

- As a consequence:

$$[\![M\ulcorner 0\urcorner]\!] = \sup_{M\ulcorner 0\urcorner \Downarrow \mathscr{D}} \mathscr{D} = \{\ulcorner 0\urcorner^{\frac{1}{2}}, \ulcorner 1\urcorner^{\frac{1}{4}}, \ulcorner 2\urcorner^{\frac{1}{8}}, \dots\}$$

Theorem

The probabilistic λ-calculus and PTMs are equiexpressive.

Probabilistic λ-Calculus: Syntax and Operational Semantics

- As an **example**, consider the following recursive program, and call it M:

```
let rec fancy x = x (+) (fancy x+1)
```

- One can easily check that:

$$M\ulcorner 0\urcorner \Downarrow \emptyset \qquad M\ulcorner 0\urcorner \Downarrow \{\ulcorner 0\urcorner^{\frac{1}{2}}\} \qquad M\ulcorner 0\urcorner \Downarrow \{\ulcorner 0\urcorner^{\frac{1}{2}}, \ulcorner 1\urcorner^{\frac{1}{4}}\} \qquad \dots$$

- As a consequence:

$$[\![M\ulcorner 0\urcorner]\!] = \sup_{M\ulcorner 0\urcorner \Downarrow \mathscr{D}} \mathscr{D} = \{\ulcorner 0\urcorner^{\frac{1}{2}}, \ulcorner 1\urcorner^{\frac{1}{4}}, \ulcorner 2\urcorner^{\frac{1}{8}}, \dots\}$$

Theorem

The probabilistic λ-calculus and PTMs are equiexpressive.

Probabilistic λ-Calculus: Syntax and Operational Semantics

- As an **example**, consider the following recursive program, and call it M:

```
let rec fancy x = x (+) (fancy x+1)
```

- One can easily check that:

$$M\ulcorner 0\urcorner \Downarrow \emptyset \qquad M\ulcorner 0\urcorner \Downarrow \{\ulcorner 0\urcorner^{\frac{1}{2}}\} \qquad M\ulcorner 0\urcorner \Downarrow \{\ulcorner 0\urcorner^{\frac{1}{2}}, \ulcorner 1\urcorner^{\frac{1}{4}}\} \qquad \dots$$

- As a consequence:

$$[\![M\ulcorner 0\urcorner]\!] = \sup_{M\ulcorner 0\urcorner \Downarrow \mathscr{D}} \mathscr{D} = \{\ulcorner 0\urcorner^{\frac{1}{2}}, \ulcorner 1\urcorner^{\frac{1}{4}}, \ulcorner 2\urcorner^{\frac{1}{8}}, \dots\}$$

Theorem

The probabilistic λ-calculus and PTMs are equiexpressive.

Probabilistic λ-Calculus: Syntax and Operational Semantics

- As an **example**, consider the following recursive program, and call it M:

```
let rec fancy x = x (+) (fancy x+1)
```

- One can easily check that:

$$M\ulcorner 0\urcorner \Downarrow \emptyset \qquad M\ulcorner 0\urcorner \Downarrow \{\ulcorner 0\urcorner^{\frac{1}{2}}\} \qquad M\ulcorner 0\urcorner \Downarrow \{\ulcorner 0\urcorner^{\frac{1}{2}}, \ulcorner 1\urcorner^{\frac{1}{4}}\} \qquad \ldots$$

- As a consequence:

$$[\![M\ulcorner 0\urcorner]\!] = \sup_{M\ulcorner 0\urcorner \Downarrow \mathscr{D}} \mathscr{D} = \{\ulcorner 0\urcorner^{\frac{1}{2}}, \ulcorner 1\urcorner^{\frac{1}{4}}, \ulcorner 2\urcorner^{\frac{1}{8}}, \ldots\}$$

Theorem

The probabilistic λ-calculus and PTMs are equiexpressive.

Probabilistic λ-Calculus: Syntax and Operational Semantics

- As an **example**, consider the following recursive program, and call it M:

```
let rec fancy x = x (+) (fancy x+1)
```

- One can easily check that:

$$M\ulcorner 0\urcorner \Downarrow \emptyset \qquad M\ulcorner 0\urcorner \Downarrow \{\ulcorner 0\urcorner^{\frac{1}{2}}\} \qquad M\ulcorner 0\urcorner \Downarrow \{\ulcorner 0\urcorner^{\frac{1}{2}}, \ulcorner 1\urcorner^{\frac{1}{4}}\} \qquad \cdots$$

- As a consequence:

$$[\![M\ulcorner 0\urcorner]\!] = \sup_{M\ulcorner 0\urcorner \Downarrow \mathscr{D}} \mathscr{D} = \{\ulcorner 0\urcorner^{\frac{1}{2}}, \ulcorner 1\urcorner^{\frac{1}{4}}, \ulcorner 2\urcorner^{\frac{1}{8}}, \ldots\}$$

Theorem

The probabilistic λ-calculus and PTMs are equiexpressive.

Probabilistic λ-Calculus: Syntax and Operational Semantics

- As an **example**, consider the following recursive program, and call it M:

```
let rec fancy x = x (+) (fancy x+1)
```

- One can easily check that:

$$M\ulcorner 0\urcorner \Downarrow \emptyset \qquad M\ulcorner 0\urcorner \Downarrow \{\ulcorner 0\urcorner^{\frac{1}{2}}\} \qquad M\ulcorner 0\urcorner \Downarrow \{\ulcorner 0\urcorner^{\frac{1}{2}}, \ulcorner 1\urcorner^{\frac{1}{4}}\} \qquad \ldots$$

- As a consequence:

$$[\![M\ulcorner 0\urcorner]\!] = \sup_{M\ulcorner 0\urcorner \Downarrow \mathscr{D}} \mathscr{D} = \{\ulcorner 0\urcorner^{\frac{1}{2}}, \ulcorner 1\urcorner^{\frac{1}{4}}, \ulcorner 2\urcorner^{\frac{1}{8}}, \ldots\}$$

Theorem

The probabilistic λ-calculus and PTMs are equiexpressive.

Probabilistic λ-Calculus: Syntax and Operational Semantics

- As an **example**, consider the following recursive program, and call it M:

```
let rec fancy x = x (+) (fancy x+1)
```

- One can easily check that:

$$M\ulcorner 0\urcorner \Downarrow \emptyset \qquad M\ulcorner 0\urcorner \Downarrow \{\ulcorner 0\urcorner^{\frac{1}{2}}\} \qquad M\ulcorner 0\urcorner \Downarrow \{\ulcorner 0\urcorner^{\frac{1}{2}}, \ulcorner 1\urcorner^{\frac{1}{4}}\} \qquad \ldots$$

- As a consequence:

$$[\![M\ulcorner 0\urcorner]\!] = \sup_{M\ulcorner 0\urcorner \Downarrow \mathscr{D}} \mathscr{D} = \{\ulcorner 0\urcorner^{\frac{1}{2}}, \ulcorner 1\urcorner^{\frac{1}{4}}, \ulcorner 2\urcorner^{\frac{1}{8}}, \ldots\}$$

Theorem

The probabilistic λ-calculus and PTMs are equiexpressive.

Section 5

Quantum λ-Calculi

A Naïve Attempt

- While looking for a quantum generalization of the λ-calculus, one could be tempted to proceed merely by enriching its syntax of **terms** as follows:

$$M, N \ ::= \ x \ \mid \ \lambda x.M \ \mid \ MN \ \mid \ r \ \mid \ \mathsf{U} \ \mid \ \texttt{new} \ \mid \ \texttt{meas}$$

  where:

  - r is a **quantum variable** pointing to the quantum store;
  - the operator `new` **creates** a new qubit;
  - the operator `meas` **measures** a qubit from the quantum store;
  - U applies a unitary transform to one (or more) qubits from the quantum store.

- This would be enough to express, e.g., some simple quantum circuits:

$$\lambda x.\lambda y.\mathsf{CNOT}\langle \mathsf{H}\ x, y\rangle \qquad\qquad \lambda x.\texttt{meas}(\mathsf{H}(\texttt{new}\ x))$$

A Naïve Attempt

$$[\emptyset, (\lambda x.\lambda y.\mathsf{CNOT}\langle \mathsf{H}\ x, y\rangle)\langle \mathtt{new}\ \ulcorner 0 \urcorner, \mathtt{new}\ \ulcorner 0 \urcorner\rangle]$$

$$\to^* [|r \to 0\rangle \otimes |q \to 0\rangle, (\lambda x.\lambda y.\mathsf{CNOT}\langle \mathsf{H}\ x, y\rangle)\langle r, q\rangle]$$

$$\to [|r \to 0\rangle \otimes |q \to 0\rangle, \mathsf{CNOT}\langle \mathsf{H}\ r, q\rangle]$$

$$\to [\frac{1}{\sqrt{2}}|r \to 0, q \to 0\rangle + \frac{1}{\sqrt{2}}|r \to 1, q \to 0\rangle, \mathsf{CNOT}\langle r, q\rangle]$$

$$\to [\frac{1}{\sqrt{2}}|r \to 0, q \to 0\rangle + \frac{1}{\sqrt{2}}|r \to 1, q \to 1\rangle, \langle r, q\rangle]$$

A Naïve Attempt

$$[\emptyset, (\lambda x.\lambda y.\mathsf{CNOT}\langle \mathsf{H}\ x, y\rangle)\langle \mathtt{new}\ \ulcorner 0 \urcorner, \mathtt{new}\ \ulcorner 0 \urcorner\rangle]$$

$$\to^* [|r \to 0\rangle \otimes |q \to 0\rangle, (\lambda x.\lambda y.\mathsf{CNOT}\langle \mathsf{H}\ x, y\rangle)\langle r, q\rangle]$$

$$\to [|r \to 0\rangle \otimes |q \to 0\rangle, \mathsf{CNOT}\langle \mathsf{H}\ r, q\rangle]$$

$$\to [\frac{1}{\sqrt{2}}|r \to 0, q \to 0\rangle + \frac{1}{\sqrt{2}}|r \to 1, q \to 0\rangle, \mathsf{CNOT}\langle r, q\rangle]$$

$$\to [\frac{1}{\sqrt{2}}|r \to 0, q \to 0\rangle + \frac{1}{\sqrt{2}}|r \to 1, q \to 1\rangle, \langle r, q\rangle]$$

A Naïve Attempt

$$[\emptyset, (\lambda x.\lambda y.\mathsf{CNOT}\langle \mathsf{H}\ x, y\rangle)\langle \mathtt{new}\ \ulcorner 0 \urcorner, \mathtt{new}\ \ulcorner 0 \urcorner\rangle]$$

$$\to^* [|r \to 0\rangle \otimes |q \to 0\rangle, (\lambda x.\lambda y.\mathsf{CNOT}\langle \mathsf{H}\ x, y\rangle)\langle r, q\rangle]$$

$$\to [|r \to 0\rangle \otimes |q \to 0\rangle, \mathsf{CNOT}\langle \mathsf{H}\ r, q\rangle]$$

$$\to [\frac{1}{\sqrt{2}}|r \to 0, q \to 0\rangle + \frac{1}{\sqrt{2}}|r \to 1, q \to 0\rangle, \mathsf{CNOT}\langle r, q\rangle]$$

$$\to [\frac{1}{\sqrt{2}}|r \to 0, q \to 0\rangle + \frac{1}{\sqrt{2}}|r \to 1, q \to 1\rangle, \langle r, q\rangle]$$

A Naïve Attempt

$$[\emptyset, (\lambda x.\lambda y.\mathsf{CNOT}\langle \mathsf{H}\ x, y\rangle)\langle \mathtt{new}\ \ulcorner 0 \urcorner, \mathtt{new}\ \ulcorner 0 \urcorner\rangle]$$

$$\to^* [|r \to 0\rangle \otimes |q \to 0\rangle, (\lambda x.\lambda y.\mathsf{CNOT}\langle \mathsf{H}\ x, y\rangle)\langle r, q\rangle]$$

$$\to [|r \to 0\rangle \otimes |q \to 0\rangle, \mathsf{CNOT}\langle \mathsf{H}\ r, q\rangle]$$

$$\to [\frac{1}{\sqrt{2}}|r \to 0, q \to 0\rangle + \frac{1}{\sqrt{2}}|r \to 1, q \to 0\rangle, \mathsf{CNOT}\langle r, q\rangle]$$

$$\to [\frac{1}{\sqrt{2}}|r \to 0, q \to 0\rangle + \frac{1}{\sqrt{2}}|r \to 1, q \to 1\rangle, \langle r, q\rangle]$$

A Naïve Attempt

$$[\emptyset, (\lambda x.\lambda y.\mathsf{CNOT}\langle \mathsf{H}\ x, y\rangle)\langle \mathtt{new}\ \ulcorner 0 \urcorner, \mathtt{new}\ \ulcorner 0 \urcorner\rangle]$$

$$\to^* [|r \to 0\rangle \otimes |q \to 0\rangle, (\lambda x.\lambda y.\mathsf{CNOT}\langle \mathsf{H}\ x, y\rangle)\langle r, q\rangle]$$

$$\to [|r \to 0\rangle \otimes |q \to 0\rangle, \mathsf{CNOT}\langle \mathsf{H}\ r, q\rangle]$$

$$\to [\frac{1}{\sqrt{2}}|r \to 0, q \to 0\rangle + \frac{1}{\sqrt{2}}|r \to 1, q \to 0\rangle, \mathsf{CNOT}\langle r, q\rangle]$$

$$\to [\frac{1}{\sqrt{2}}|r \to 0, q \to 0\rangle + \frac{1}{\sqrt{2}}|r \to 1, q \to 1\rangle, \langle r, q\rangle]$$

A Naïve Attempt

- This approach has some **fundamental problems**, however.
- Take the term $(\lambda x.\mathsf{CNOT}\langle x, x\rangle)\texttt{new}$ and suppose, as in the previous example, to reduce it in CBV:

$$[\emptyset, (\lambda x.\mathsf{CNOT}\langle x, x\rangle)(\texttt{new}\ulcorner 0\urcorner)]$$
$$\to [|r \to 0\rangle, (\lambda x.\mathsf{CNOT}\langle x, x\rangle)r]$$
$$\to [|r \to 0\rangle, \mathsf{CNOT}\langle r, r\rangle]$$

- Qubits can be freely duplicated or erased!
  - And this goes against the principles of quantum mechanics.
- We need a way to force qubits to be used **exactly** once when passed to functions.

A Naïve Attempt

- This approach has some **fundamental problems**, however.
- Take the term $(\lambda x.\mathsf{CNOT}\langle x, x\rangle)\mathtt{new}$ and suppose, as in the previous example, to reduce it in CBV:

$$[\emptyset, (\lambda x.\mathsf{CNOT}\langle x, x\rangle)(\mathtt{new}\ulcorner 0\urcorner)]$$
$$\to [|r \to 0\rangle, (\lambda x.\mathsf{CNOT}\langle x, x\rangle)r]$$
$$\to [|r \to 0\rangle, \mathsf{CNOT}\langle r, r\rangle]$$

- Qubits can be freely duplicated or erased!
  - And this goes against the principles of quantum mechanics.
- We need a way to force qubits to be used **exactly** once when passed to functions.

A Naïve Attempt

- This approach has some **fundamental problems**, however.
- Take the term $(\lambda x.\mathsf{CNOT}\langle x, x\rangle)\mathtt{new}$ and suppose, as in the previous example, to reduce it in CBV:

$$[\emptyset, (\lambda x.\mathsf{CNOT}\langle x, x\rangle)(\mathtt{new}\ulcorner 0\urcorner)]$$
$$\to [|r \to 0\rangle, (\lambda x.\mathsf{CNOT}\langle x, x\rangle)r]$$
$$\to [|r \to 0\rangle, \mathsf{CNOT}\langle r, r\rangle]$$

- Qubits can be freely duplicated or erased!
  - And this goes against the principles of quantum mechanics.
- We need a way to force qubits to be used **exactly** once when passed to functions.

A Naïve Attempt

- This approach has some **fundamental problems**, however.
- Take the term $(\lambda x.\mathsf{CNOT}\langle x, x\rangle)\mathtt{new}$ and suppose, as in the previous example, to reduce it in CBV:

$$[\emptyset, (\lambda x.\mathsf{CNOT}\langle x, x\rangle)(\mathtt{new}\ulcorner 0\urcorner)]$$
$$\to [|r \to 0\rangle, (\lambda x.\mathsf{CNOT}\langle x, x\rangle)r]$$
$$\to [|r \to 0\rangle, \mathsf{CNOT}\langle r, r\rangle]$$

- Qubits can be freely duplicated or erased!
  - And this goes against the principles of quantum mechanics.
- We need a way to force qubits to be used **exactly** once when passed to functions.

Linearity and the λ-Calculus

- First of all, let us impose that in any abstraction $\lambda x.M$, the variable x occurs **exactly** once in M.
  - Copying and erasure **not** possible, anymore.
  - But the calculus **loses** much of its expressive power.
- The idea is to **refine** the calculus in such a way as to reintroduce erasure and copying, but in a controlled way.
  - We mark as $!M$ all those subterms which can be (potentially) copied or erased;
  - Besides the usual linear abstraction $\lambda x.M$, there is a non-linear abstraction $\lambda !x.M$.
- Altogether, then, the language of terms becomes:

$$M, N ::= x \mid \lambda x.M \mid \lambda !x.M \mid MN \mid !M$$

$$r \mid \mathsf{U} \mid \texttt{new} \mid \texttt{meas}$$

where we insist that any term in the form $!M$ does not contain any quantum variable.

Linearity and the λ-Calculus

- First of all, let us impose that in any abstraction $\lambda x.M$, the variable x occurs **exactly** once in M.
  - Copying and erasure **not** possible, anymore.
  - But the calculus **loses** much of its expressive power.
- The idea is to **refine** the calculus in such a way as to reintroduce erasure and copying, but in a controlled way.
  - We mark as $!M$ all those subterms which can be (potentially) copied or erased;
  - Besides the usual linear abstraction $\lambda x.M$, there is a non-linear abstraction $\lambda !x.M$.
- Altogether, then, the language of terms becomes:

$$M, N \ ::= x \ \Big| \ \lambda x.M \ \Big| \ \lambda !x.M \ \Big| \ MN \ \Big| \ !M$$
$$r \ \Big| \ \mathsf{U} \ \Big| \ \texttt{new} \ \Big| \ \texttt{meas}$$

where we insist that any term in the form $!M$ does not contain any quantum variable.

Operational Semantics

Evaluation Contexts

$$E ::= [\cdot] \mid EM \mid ME \mid \lambda x.E \mid \lambda !x.E$$

Small Step Rules

$$[\mathcal{Q}, E[(\lambda x.M)N]] \Rightarrow \{[\mathcal{Q}, E[M\{x/N\}]]^1\}$$

$$[\mathcal{Q}, E[(\lambda !x.M)!N]] \Rightarrow \{[\mathcal{Q}, E[M\{x/N\}]]^1\}$$

$$[\mathcal{Q}, E[\texttt{meas}\ r]] \Rightarrow \{[\Pi_r^0(\mathcal{Q}), E[0]]^{\Xi_r^0(\mathcal{Q})}, [\Pi_r^1(\mathcal{Q}), E[1]]^{\Xi_r^1(\mathcal{Q})}\}$$

$$[\mathcal{Q}, E[\cup\ r]] \Rightarrow \{[\cup_r(\mathcal{Q}), E[r]]^1\}$$

$$[\mathcal{Q}, E[\texttt{new}\ 0]] \Rightarrow \{[\mathcal{Q} \otimes |r \to 0\rangle, E[r]]^1\}$$

$$[\mathcal{Q}, E[\texttt{new}\ 1]] \Rightarrow \{[\mathcal{Q} \otimes |r \to 1\rangle, E[r]]^1\}$$

Operational Semantics

Evaluation Contexts

$$E ::= [\cdot] \mid EM \mid ME \mid \lambda x.E \mid \lambda !x.E$$

Small Step Rules

$$[\mathcal{Q}, E[(\lambda x.M)N]] \Rightarrow \{[\mathcal{Q}, E[M\{x/N\}]]^1\}$$

$$[\mathcal{Q}, E[(\lambda !x.M)!N]] \Rightarrow \{[\mathcal{Q}, E[M\{x/N\}]]^1\}$$

$$[\mathcal{Q}, E[\texttt{meas}\ r]] \Rightarrow \{[\Pi_r^0(\mathcal{Q}), E[0]]^{\Xi_r^0(\mathcal{Q})}, [\Pi_r^1(\mathcal{Q}), E[1]]^{\Xi_r^1(\mathcal{Q})}\}$$

$$[\mathcal{Q}, E[\cup\ r]] \Rightarrow \{[\cup_r(\mathcal{Q}), E[r]]^1\}$$

$$[\mathcal{Q}, E[\texttt{new}\ 0]] \Rightarrow \{[\mathcal{Q} \otimes |r \to 0\rangle, E[r]]^1\}$$

$$[\mathcal{Q}, E[\texttt{new}\ 1]] \Rightarrow \{[\mathcal{Q} \otimes |r \to 1\rangle, E[r]]^1\}$$

Operational Semantics

Evaluation Contexts

$$E ::= [\cdot] \mid EM \mid ME \mid \lambda x.E \mid \lambda !x.E$$

Small Step Rules

$$[\mathcal{Q}, E[(\lambda x.M)N]] \Rightarrow \{[\mathcal{Q}, E[M\{x/N\}]]^1\}$$

$$[\mathcal{Q}, E[(\lambda !x.M)!N]] \Rightarrow \{[\mathcal{Q}, E[M\{x/N\}]]^1\}$$

$$[\mathcal{Q}, E[\texttt{meas}\ r]] \Rightarrow \{[\Pi_r^0(\mathcal{Q}), E[0]]^{\Xi_r^0(\mathcal{Q})}, [\Pi_r^1(\mathcal{Q}), E[1]]^{\Xi_r^1(\mathcal{Q})}\}$$

$$[\mathcal{Q}, E[\cup\ r]] \Rightarrow \{[\cup_r(\mathcal{Q}), E[r]]^1\}$$

$$[\mathcal{Q}, E[\texttt{new}\ 0]] \Rightarrow \{[\mathcal{Q} \otimes |r \to 0\rangle, E[r]]^1\}$$

$$[\mathcal{Q}, E[\texttt{new}\ 1]] \Rightarrow \{[\mathcal{Q} \otimes |r \to 1\rangle, E[r]]^1\}$$

Operational Semantics

Evaluation Contexts

$$E ::= [\cdot] \mid EM \mid ME \mid \lambda x.E \mid \lambda !x.E$$

Small Step Rules

$$[\mathcal{Q}, E[(\lambda x.M)N]] \Rightarrow \{[\mathcal{Q}, E[M\{x/N\}]]^1\}$$

$$[\mathcal{Q}, E[(\lambda !x.M)!N]] \Rightarrow \{[\mathcal{Q}, E[M\{x/N\}]]^1\}$$

$$[\mathcal{Q}, E[\texttt{meas}\ r]] \Rightarrow \{[\Pi_r^0(\mathcal{Q}), E[0]]^{\Xi_r^0(\mathcal{Q})}, [\Pi_r^1(\mathcal{Q}), E[1]]^{\Xi_r^1(\mathcal{Q})}\}$$

$$[\mathcal{Q}, E[\mathsf{U}\ r]] \Rightarrow \{[\mathsf{U}_r(\mathcal{Q}), E[r]]^1\}$$

$$[\mathcal{Q}, E[\texttt{new}\ 0]] \Rightarrow \{[\mathcal{Q} \otimes |r \to 0\rangle, E[r]]^1\}$$

$$[\mathcal{Q}, E[\texttt{new}\ 1]] \Rightarrow \{[\mathcal{Q} \otimes |r \to 1\rangle, E[r]]^1\}$$

Operational Semantics

Evaluation Contexts

$$E ::= [\cdot] \mid EM \mid ME \mid \lambda x.E \mid \lambda !x.E$$

Small Step Rules

$$\begin{aligned}
[\mathcal{Q}, E[(\lambda x.M)N]] &\Rightarrow \{[\mathcal{Q}, E[M\{x/N\}]]^1\} \\
[\mathcal{Q}, E[(\lambda !x.M)!N]] &\Rightarrow \{[\mathcal{Q}, E[M\{x/N\}]]^1\} \\
[\mathcal{Q}, E[\texttt{meas}\ r]] &\Rightarrow \{[\Pi_r^0(\mathcal{Q}), E[0]]^{\Xi_r^0(\mathcal{Q})}, [\Pi_r^1(\mathcal{Q}), E[1]]^{\Xi_r^1(\mathcal{Q})}\} \\
[\mathcal{Q}, E[\mathsf{U}\ r]] &\Rightarrow \{[\mathsf{U}_r(\mathcal{Q}), E[r]]^1\} \\
[\mathcal{Q}, E[\texttt{new}\ 0]] &\Rightarrow \{[\mathcal{Q} \otimes |r \to 0\rangle, E[r]]^1\} \\
[\mathcal{Q}, E[\texttt{new}\ 1]] &\Rightarrow \{[\mathcal{Q} \otimes |r \to 1\rangle, E[r]]^1\}
\end{aligned}$$

Expressive Power

Theorem

Any uniform quantum circuit family $\{C_n\}_{n\in\mathbb{N}}$ *is simulated by a* `meas`*-free term* M*.*

- Given s, the term M computes a *code* for $C_{|s|}$
- Then, it applies $C_{|s|}$ to s.

Theorem

Any `meas`*-free term* M *computes the same function as a uniform quantum circuit family* $\{C_n\}_{n\in\mathbb{N}}$*.*

- A term M can be evaluated by first reducing all the "classical" redexes, and the reducing the "quantum" ones.
- This is possible due to a standardization result.

Expressive Power

Theorem

Any uniform quantum circuit family $\{C_n\}_{n\in\mathbb{N}}$ *is simulated by a* `meas`*-free term* M*.*

- Given s, the term M computes a *code* for $C_{|s|}$
- Then, it applies $C_{|s|}$ to s.

Theorem

Any `meas`*-free term* M *computes the same function as a uniform quantum circuit family* $\{C_n\}_{n\in\mathbb{N}}$*.*

- A term M can be evaluated by first reducing all the "classical" redexes, and the reducing the "quantum" ones.
- This is possible due to a standardization result.

Expressive Power

Theorem

Any uniform quantum circuit family $\{C_n\}_{n\in\mathbb{N}}$ *is simulated by a* `meas`*-free term* M*.*

- Given s, the term M computes a *code* for $C_{|s|}$
- Then, it applies $C_{|s|}$ to s.

Theorem

Any `meas`*-free term* M *computes the same function as a uniform quantum circuit family* $\{C_n\}_{n\in\mathbb{N}}$*.*

- A term M can be evaluated by first reducing all the "classical" redexes, and the reducing the "quantum" ones.
- This is possible due to a standardization result.

Expressive Power

Theorem

Any uniform quantum circuit family $\{C_n\}_{n\in\mathbb{N}}$ *is simulated by a* `meas`*-free term* M*.*

- Given s, the term M computes a *code* for $C_{|s|}$
- Then, it applies $C_{|s|}$ to s.

Theorem

Any `meas`*-free term* M *computes the same function as a uniform quantum circuit family* $\{C_n\}_{n\in\mathbb{N}}$*.*

- A term M can be evaluated by first reducing all the "classical" redexes, and the reducing the "quantum" ones.
- This is possible due to a standardization result.

Wrapping Up

- Probabilistic and quantum programming are both at their infancy.
  - A lot of **interest**.
  - Many different **proposals**.
  - Not so many results **relating** one programming language to the other.
- We have only presented *some* of the many proposals for models and calculi.
- Very interesting topics we have no time to talk about:
  - **Denotational** semantics.
  - Type **Systems**.
  - Implicit Computational Complexity.
  - **Concrete** programming languages: Quipper, Church, Venture, QScript.

Wrapping Up

- Probabilistic and quantum programming are both at their infancy.
  - A lot of **interest**.
  - Many different **proposals**.
  - Not so many results **relating** one programming language to the other.
- We have only presented *some* of the many proposals for models and calculi.
- Very interesting topics we have no time to talk about:
  - **Denotational** semantics.
  - Type **Systems**.
  - Implicit Computational Complexity.
  - **Concrete** programming languages: Quipper, Church, Venture, QScript.

Wrapping Up

- Probabilistic and quantum programming are both at their infancy.
  - A lot of **interest**.
  - Many different **proposals**.
  - Not so many results **relating** one programming language to the other.
- We have only presented *some* of the many proposals for models and calculi.
- Very interesting topics we have no time to talk about:
  - **Denotational** semantics.
  - Type **Systems**.
  - Implicit Computational Complexity.
  - **Concrete** programming languages: Quipper, Church, Venture, QScript.

Thank You!

Questions?