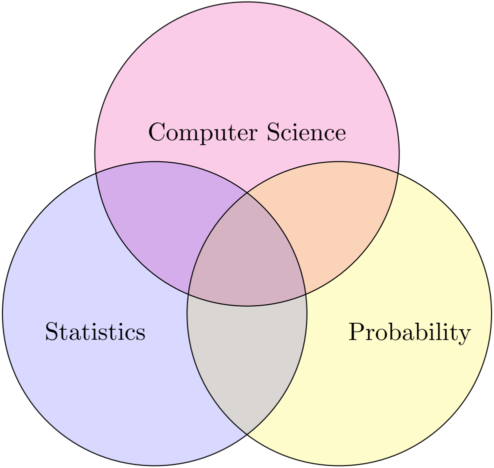
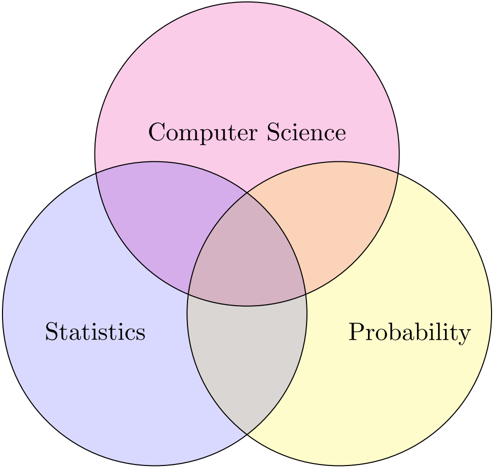
Conditional Independence, Computability, and Measurability

Daniel M. Roy

Research Fellow
Emmanuel College
University of Cambridge

MFPS XXX, Cornell University, Ithaca, June 14, 2014


Computer Science
Statistics
Probability



Algorithmic processes that describe and transform uncertainty.

The stochastic inference problem (informal version)

The stochastic inference problem (informal version)

INPUT: `guesser` and `checker` probabilistic programs.

The stochastic inference problem (informal version)

INPUT: `guesser` and `checker` probabilistic programs.

OUTPUT: a sample from the same distribution as the program

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

The stochastic inference problem (informal version)

INPUT: `guesser` and `checker` probabilistic programs.

OUTPUT: a sample from the same distribution as the program

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

This computation captures **Bayesian statistical inference**.

The stochastic inference problem (informal version)

INPUT: `guesser` and `checker` probabilistic programs.

OUTPUT: a sample from the same distribution as the program

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

This computation captures **Bayesian statistical inference**.

"prior" distribution $\longleftrightarrow$ distribution of `guesser()`

"likelihood(g)" $\longleftrightarrow$ $\Pr\big($ `checker(`g`)` is `True` $\big)$

"posterior" distribution $\longleftrightarrow$ distribution of return value

Example: predicting next coin toss in a sequence

Example: predicting next coin toss in a sequence

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

Example: predicting next coin toss in a sequence

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

Let $n \geq 0$ and $x_1, \ldots, x_n \in \{0, 1\}$. E.g., $0, 0, 1, 0, 0, 0, ?$

Example: predicting next coin toss in a sequence

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

Let $n \geq 0$ and $x_1, \ldots, x_n \in \{0, 1\}$. E.g., $0, 0, 1, 0, 0, 0, ?$

`guesser()`:

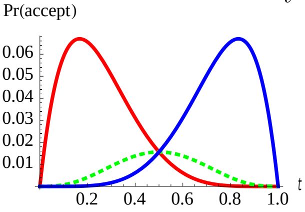
- *sample* θ and U independently and uniformly in $[0, 1]$, and
- *return* (θ, X) where $X = 1(U \leq \theta)$.

`checker(`θ`,`x`)`:

- *sample* $U_1, \ldots, U_n$ independently and uniformly in $[0, 1]$,
- *let* $X_i = 1(U_i \leq \theta)$, and
- *accept* if and only if $X_i = x_i$ for all i.

Let $s = x_1 + \cdots + x_n$ and let U be uniformly distributed.
For all $t \in [0,1]$, we have $\Pr\big(U \le t\big) = t$ and

$$\begin{aligned}\Pr\big(\texttt{checker}(t,x) \text{ is } \texttt{True}\big) &= \Pr\big(\ \forall i\ (\ U_i \le t \iff x_i = 1\ )\big) \\ &= t^s(1-t)^{n-s}.\end{aligned}$$



$n = 6,\ s \in \{1,3,5\}.$

$$\Pr\big(\texttt{checker}(U,x) \text{ is } \texttt{True}\big) = \int_0^1 t^s(1-t)^{n-s}\,\mathrm{d}t = \frac{(s)!(n-s)!}{(n+1)!} =: Z(s)$$

Let $p(t)\mathrm{d}t$ be the probability that the accepted $\theta \in [t, t+\mathrm{d}t)$.

$$p(t)\mathrm{d}t \approx t^s(1-t)^{n-s}\mathrm{d}t + \big(1 - Z(s)\big)p(t)\mathrm{d}t \approx \frac{t^s(1-t)^{n-s}}{Z(s)}\mathrm{d}t$$

Probability that the accepted $X = 1$ is then $\int t\,p(t)\mathrm{d}t = \frac{s+1}{n+2}$.

Example: fitting a line to data (aka linear regression)

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

Let $(x_i, y_i) \in \mathbb{R}^2$ and $\sigma, \nu, \varepsilon > 0$.

`guesser()`:

- *sample* coefficients α, β independently from $\text{Normal}(0, \sigma^2)$.

`checker(`α`,`β`)`:

- *sample* independent noise variables ξ_i from $\text{Normal}(0, \nu^2)$,
- *let* $F(x) = \alpha x + \beta$ and $Y_i = F(x_i) + \xi_i$, and
- *accept* if and only if $|Y_i - y_i| < \varepsilon$ for all i.

Example: fitting a line to data (aka linear regression)

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

Let $(x_i, y_i) \in \mathbb{R}^2$ and $\sigma, \nu, \varepsilon > 0$.

`guesser()`:

- *sample* coefficients α, β independently from $\text{Normal}(0, \sigma^2)$.

`checker(`α`,`β`)`:

- *sample* independent noise variables ξ_i from $\text{Normal}(0, \nu^2)$,
- *let* $F(x) = \alpha x + \beta$ and $Y_i = F(x_i) + \xi_i$, and
- *accept* if and only if $|Y_i - y_i| < \varepsilon$ for all i.

Note that $\varepsilon = 0$ doesn't work, but the limit $\varepsilon \to 0$ makes sense.

Fantasy example: extracting 3D structure from images

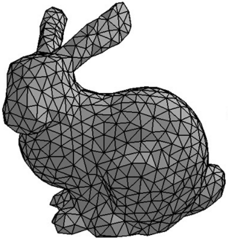
```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

Fantasy example: extracting 3D structure from images

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

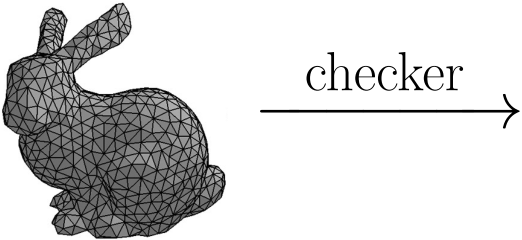
Fantasy example: extracting 3D structure from images

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```



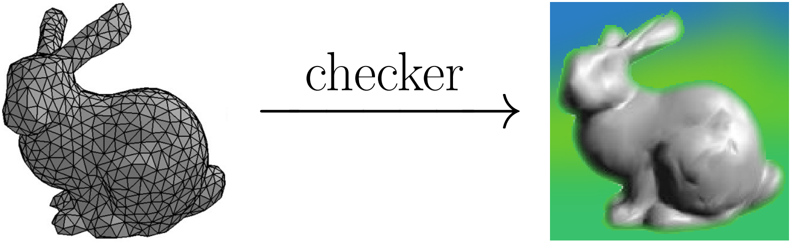
Fantasy example: extracting 3D structure from images

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```



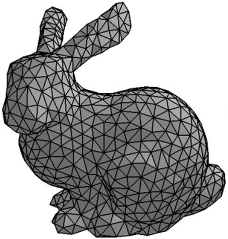
Fantasy example: extracting 3D structure from images

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```



Fantasy example: extracting 3D structure from images

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```



checker →

Fantasy example: extracting 3D structure from images

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

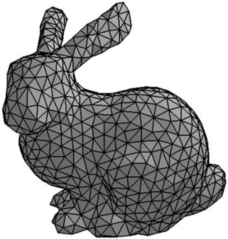
Fantasy example: extracting 3D structure from images

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

$\xleftarrow{\text{inference}}$

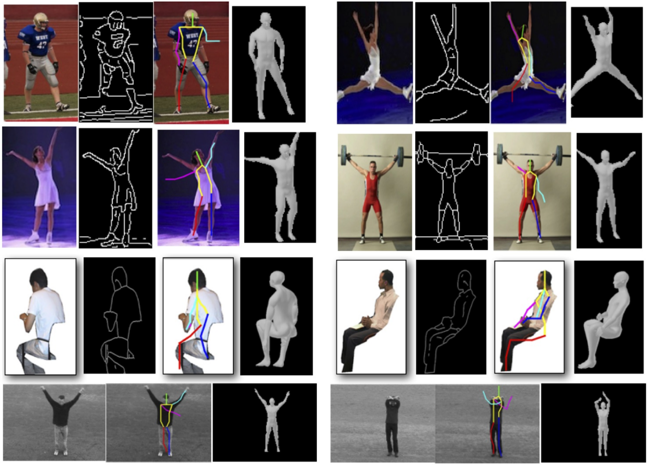
Fantasy example: extracting 3D structure from images

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```



← inference

Example: not so fantastical [Mansinghka et al.]



The stochastic inference problem

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

The stochastic inference problem

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

Let U be a Uniform$(0, 1)$ random variable.

The stochastic inference problem

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

Let U be a Uniform$(0, 1)$ random variable.
Let S and T be a computable metric space.

The stochastic inference problem

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

Let U be a Uniform$(0,1)$ random variable.
Let S and T be a computable metric space.

INPUT:
$X : [0,1] \to S$,
$Y : [0,1] \to T$, and
$x \in S$.

The stochastic inference problem

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

Let U be a Uniform$(0, 1)$ random variable.
Let S and T be a computable metric space.

INPUT:
$X : [0, 1] \to S$,
$Y : [0, 1] \to T$, and
$x \in S$.

OUTPUT:
a sample from $\Pr\big(Y(U)|X(U) = x\big)$,
i.e., the conditional distribution of $Y(U)$ given $X(U) = x$.

Bayesian statistics

1. Express statistical assumptions via **probability distributions**.

$$\underbrace{\Pr(\text{parameters}, \text{data})}_{\text{joint}} = \underbrace{\Pr(\text{parameters})}_{\text{prior}} \underbrace{\Pr(\text{data} \mid \text{parameters})}_{\text{model/likelihood}}$$

2. Statistical inference from data $\to$ parameters via **conditioning**.

$$\Pr(\text{parameters}, \text{data}),\ x \xmapsto{\text{conditioning}} \underbrace{\Pr(\text{parameters} \mid \text{data} = x)}_{\text{posterior}}$$

Probabilistic programming

1. Represent probability distributions by ~~*formulas*~~ **probabilistic programs *that generate samples***.
2. Build **generic algorithms for probabilistic conditioning** using probabilistic programs as representations.

Talk Outline

1. **The stochastic inference problem**
2. Where are we now in probabilistic programming?
3. Approximability and Exchangeability:
   When can we represent conditional independence?
4. Conclusion

Talk Outline

1. The stochastic inference problem
2. **Where are we now in probabilistic programming?**
3. Approximability and Exchangeability:
   When can we represent conditional independence?
4. Conclusion

MIT-Church, Venture (MIT), webchurch (Stanford), BUGS, Tabular (MSR) Stan (Columbia), BLOG (Berkeley), Infer.NET (MSR), Figaro (CRA), FACTORIE (UMass), ProbLog (KU Leuven), HANSEI (Indiana), . . .

DARPA PPAML

PROBABILISTIC PROGRAMMING FOR ADVANCED MACHINE LEARNING

MIT-Church, Venture (MIT), webchurch (Stanford), BUGS, Tabular (MSR) Stan (Columbia), BLOG (Berkeley), Infer.NET (MSR), Figaro (CRA), FACTORIE (UMass), ProbLog (KU Leuven), HANSEI (Indiana), ...

DARPA PPAML

PROBABILISTIC PROGRAMMING FOR ADVANCED MACHINE LEARNING

Questions raised

MIT-Church, Venture (MIT), webchurch (Stanford), BUGS, Tabular (MSR) Stan (Columbia), BLOG (Berkeley), Infer.NET (MSR), Figaro (CRA), FACTORIE (UMass), ProbLog (KU Leuven), HANSEI (Indiana), . . .

DARPA PPAML
PROBABILISTIC PROGRAMMING FOR ADVANCED MACHINE LEARNING

Questions raised

- Which operations in probability theory can we perform when distributions are represented by programs?

MIT-Church, Venture (MIT), webchurch (Stanford), BUGS, Tabular (MSR)
Stan (Columbia), BLOG (Berkeley), Infer.NET (MSR), Figaro (CRA),
FACTORIE (UMass), ProbLog (KU Leuven), HANSEI (Indiana), . . .

DARPA PPAML
PROBABILISTIC PROGRAMMING FOR ADVANCED MACHINE LEARNING

Questions raised

- Which operations in probability theory can we perform when distributions are represented by programs?
- When can we perform these computations efficiently?

MIT-Church, Venture (MIT), webchurch (Stanford), BUGS, Tabular (MSR) Stan (Columbia), BLOG (Berkeley), Infer.NET (MSR), Figaro (CRA), FACTORIE (UMass), ProbLog (KU Leuven), HANSEI (Indiana), . . .

DARPA

PPAML

PROBABILISTIC PROGRAMMING FOR ADVANCED MACHINE LEARNING

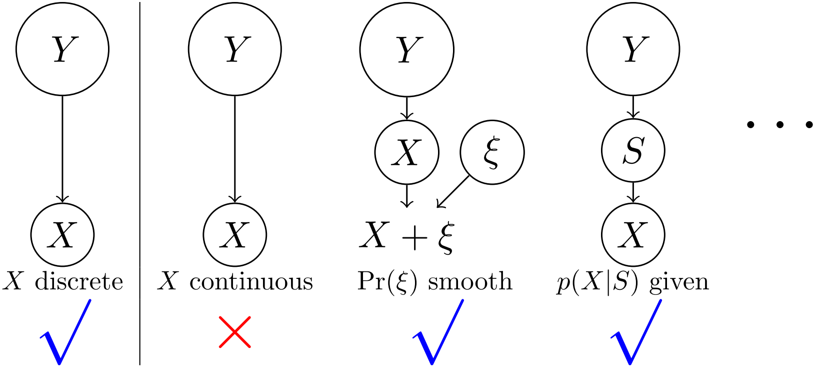
Questions raised

- Which operations in probability theory can we perform when distributions are represented by programs?
- When can we perform these computations efficiently?
- How are statistical properties (e.g., symmetries) of a distribution reflected in the structure of the computation representing it?

Q: Can we automate conditioning?

$$\Pr(X,Y),\ x \longmapsto \Pr(Y|X=x)$$

A: No, but almost.

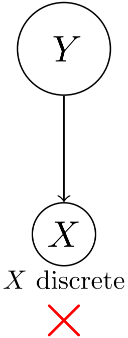


[Freer and **R.**, 2010] [Ackerman, Freer, and **R.**, 2011] ...

Q: What about EFFICIENT inference?

$$\Pr(X, Y),\ x \longmapsto \Pr(Y | X = x)$$

A: It's complicated.

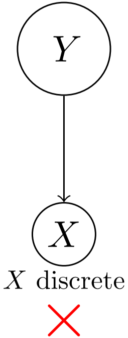



```
def hash_of_random_string(n):
    str = random_binary_string(n)
    return cryptographic_hash(str)
```

Q: What about EFFICIENT inference?

$$\Pr(X, Y),\ x \longmapsto \Pr(Y|X = x)$$

A: It's complicated.



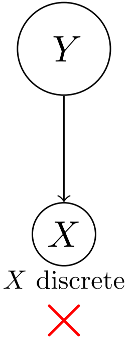

```
def hash_of_random_string(n):
    str = random_binary_string(n)
    return cryptographic_hash(str)
```

Q: What explains the success of probabilistic methods?

Q: What about EFFICIENT inference?

$$\Pr(X, Y),\ x \longmapsto \Pr(Y|X = x)$$

A: It's complicated.




```
def hash_of_random_string(n):
    str = random_binary_string(n)
    return cryptographic_hash(str)
```

Q: What explains the success of probabilistic methods?
A: Structure like conditional independence.

- Bayes nets are representations of distributions that expose conditional independence structure via a directed graph.
- The complexity of exact inference in Bayes nets is controlled by the the *tree width* of the graph.

Q: Are probabilistic programs sufficiently general as representations for stochastic processes?

We are missing a notion of approximation!

Theorem (Avigad, Freer, R., and Rute).
"Approximate samplers can represent conditional independencies that exact samplers cannot."

Talk Outline

1. The stochastic inference problem
2. Where are we now in probabilistic programming?
3. Approximability and Exchangeability:
   When can we represent conditional independence?
4. Conclusion

Talk Outline

1. The stochastic inference problem
2. Where are we now in probabilistic programming?
3. **Approximability and Exchangeability: When can we represent conditional independence?**
4. Conclusion

Concrete example of an exchangeable sequence

```
Sum = 1.0; Total = 2.0
def next_draw():
    global Sum, Total
    y = bernoulli(Sum/Total)
    Sum += y; Total += 1
    return y
```

Concrete example of an exchangeable sequence

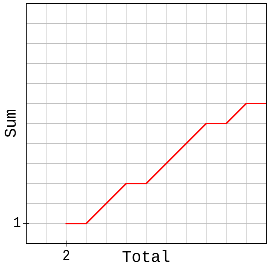
```
Sum = 1.0; Total = 2.0
def next_draw():
    global Sum, Total
    y = bernoulli(Sum/Total)
    Sum += y; Total += 1
    return y
```

```
>>> repeat(next_draw, 10)
[0, 1, 1, 0, 1, 1, 1, 0, 1, 0]
```

Concrete example of an exchangeable sequence

```
Sum = 1.0; Total = 2.0
def next_draw():
    global Sum, Total
    y = bernoulli(Sum/Total)
    Sum += y; Total += 1
    return y
```

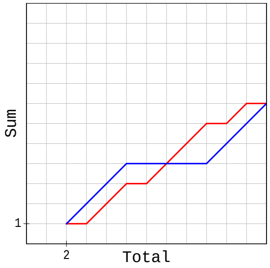
```
>>> repeat(next_draw, 10)
[0, 1, 1, 0, 1, 1, 1, 0, 1, 0]
```



Concrete example of an exchangeable sequence

```
Sum = 1.0; Total = 2.0
def next_draw():
    global Sum, Total
    y = bernoulli(Sum/Total)
    Sum += y; Total += 1
    return y
```

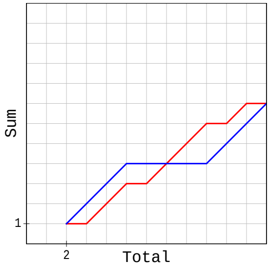
```
>>> repeat(next_draw, 10)
[0, 1, 1, 0, 1, 1, 1, 0, 1, 0]
```

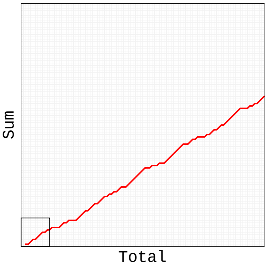


Concrete example of an exchangeable sequence

```
Sum = 1.0; Total = 2.0
def next_draw():
    global Sum, Total
    y = bernoulli(Sum/Total)
    Sum += y; Total += 1
    return y
```

```
>>> repeat(next_draw, 10)
[0, 1, 1, 0, 1, 1, 1, 0, 1, 0]
```

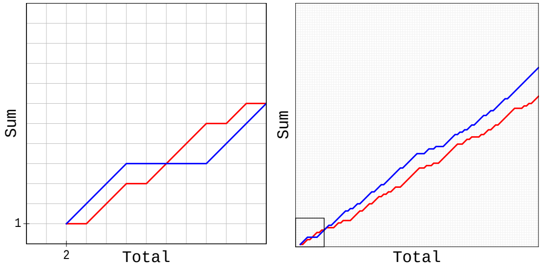




Concrete example of an exchangeable sequence

```
Sum = 1.0; Total = 2.0
def next_draw():
    global Sum, Total
    y = bernoulli(Sum/Total)
    Sum += y; Total += 1
    return y
```

```
>>> repeat(next_draw, 10)
[0, 1, 1, 0, 1, 1, 1, 0, 1, 0]
```

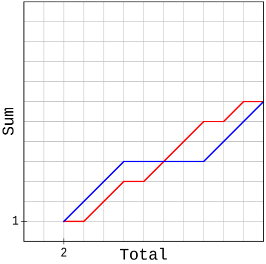


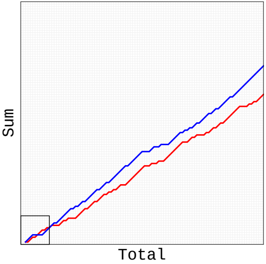
Concrete example of an exchangeable sequence

```
Sum = 1.0; Total = 2.0
def next_draw():
    global Sum, Total
    y = bernoulli(Sum/Total)
    Sum += y; Total += 1
    return y
```

```
theta = uniform(0,1)
def next_draw():
   return bernoulli(theta)
```

```
>>> repeat(next_draw, 10)
[0, 1, 1, 0, 1, 1, 1, 0, 1, 0]
```





Exchangeability and Conditional Independence

Definition. A sequence $Y = (Y_1, Y_2, \dots)$ of random variables is **exchangeable** when

$$(Y_1, \dots, Y_n) \stackrel{d}{=} (Y_{\pi(1)}, \dots, Y_{\pi(n)}), \tag{1}$$

for all $n \in \mathbb{N}$ and permutation π of $\{1, \dots, n\}$.

Exchangeability and Conditional Independence

Definition. A sequence $Y = (Y_1, Y_2, \dots)$ of random variables is **exchangeable** when

$$(Y_1, \dots, Y_n) \overset{d}{=} (Y_{\pi(1)}, \dots, Y_{\pi(n)}), \tag{1}$$

for all $n \in \mathbb{N}$ and permutation π of $\{1, \dots, n\}$.

Theorem (de Finetti). *The following are equivalent:*

1. $(Y_1, Y_2, \dots)$ *is* **exchangeable***;*

Exchangeability and Conditional Independence

Definition. A sequence $Y = (Y_1, Y_2, \dots)$ of random variables is **exchangeable** when

$$(Y_1, \dots, Y_n) \overset{d}{=} (Y_{\pi(1)}, \dots, Y_{\pi(n)}), \qquad (1)$$

for all $n \in \mathbb{N}$ and permutation π of $\{1, \dots, n\}$.

Theorem (de Finetti). *The following are equivalent:*

1. $(Y_1, Y_2, \dots)$ *is* **exchangeable***;*
2. $(Y_1, Y_2, \dots)$ *is* **conditionally i.i.d. given some** θ*;*

Exchangeability and Conditional Independence

Definition. A sequence $Y = (Y_1, Y_2, \dots)$ of random variables is **exchangeable** when

$$(Y_1, \dots, Y_n) \stackrel{d}{=} (Y_{\pi(1)}, \dots, Y_{\pi(n)}), \tag{1}$$

for all $n \in \mathbb{N}$ and permutation π of $\{1, \dots, n\}$.

Theorem (de Finetti). *The following are equivalent:*

1. $(Y_1, Y_2, \dots)$ *is* **exchangeable***;*
2. $(Y_1, Y_2, \dots)$ *is* **conditionally i.i.d. given some** θ*;*
3. *Exists* f *such that*

$$(Y_1, Y_2, Y_3, \dots) \stackrel{d}{=} (f(\theta, U_1), f(\theta, U_2), f(\theta, U_3), \dots) \tag{2}$$

for i.i.d. uniform $\theta, U_1, U_2, \dots$.

Exchangeability and Conditional Independence

Definition. A sequence $Y = (Y_1, Y_2, \dots)$ of random variables is **exchangeable** when

$$(Y_1, \dots, Y_n) \overset{d}{=} (Y_{\pi(1)}, \dots, Y_{\pi(n)}), \tag{1}$$

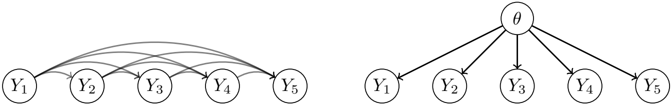
for all $n \in \mathbb{N}$ and permutation π of $\{1, \dots, n\}$.

Theorem (de Finetti). *The following are equivalent:*

1. $(Y_1, Y_2, \dots)$ *is* **exchangeable***;*
2. $(Y_1, Y_2, \dots)$ *is* **conditionally i.i.d. given some** θ*;*
3. *Exists* f *such that*

$$(Y_1, Y_2, Y_3, \dots) \overset{d}{=} (f(\theta, U_1), f(\theta, U_2), f(\theta, U_3), \dots) \tag{2}$$

for i.i.d. uniform $\theta, U_1, U_2, \dots$.

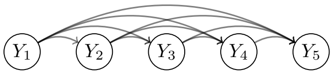


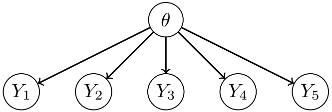
Informally: using f, we can sample Y_i's in parallel.

We can extract the hidden parallelism. [Freer and **R.**, 2012]

```
Sum = 1.0; Total = 2.0
def next_draw():
    global Sum, Total
    y = bernoulli(Sum/Total)
    Sum += y; Total += 1
    return y
```

```
theta = uniform(0,1)
def next_draw():
   return bernoulli(theta)
```

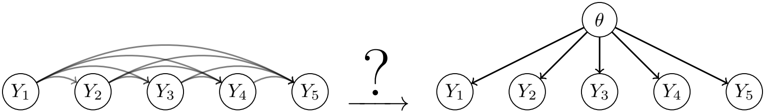




We can extract the hidden parallelism. [Freer and **R.**, 2012]

```
Sum = 1.0; Total = 2.0
def next_draw():
    global Sum, Total
    y = bernoulli(Sum/Total)
    Sum += y; Total += 1
    return y
```

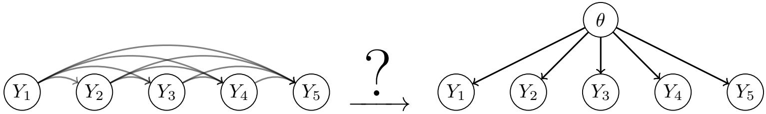
```
theta = uniform(0,1)
def next_draw():
   return bernoulli(theta)
```



We can extract the hidden parallelism. [Freer and **R.**, 2012]

```
Sum = 1.0; Total = 2.0
def next_draw():
    global Sum, Total
    y = bernoulli(Sum/Total)
    Sum += y; Total += 1
    return y
```

```
theta = uniform(0,1)
def next_draw():
   return bernoulli(theta)
```



Theorem (Freer and R., 2012). *The distribution of an exchangeable sequence Y is computable if and only if there is an almost computable f such that* $(Y_1, Y_2, \dots) \stackrel{d}{=} (f(\theta, U_1), f(\theta, U_2), \dots)$.

We can always recover hidden parallel structure, exposing conditional independence to the inference engine.

Where else can we find
hidden conditional independence?

Can we extract it for inference?

Exchangeable arrays in models of graph/relational data

Definition.

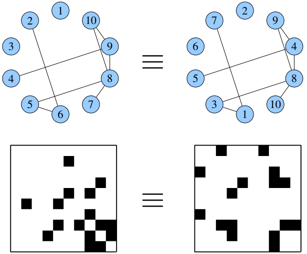
structure	*symmetry*	*definition*
sequence (Y_n)	**exchangeable**	$(Y_n) \overset{d}{=} (Y_{\pi(n)})$
array $(X_{i,j})$	**separately exchangeable**	$(X_{i,j}) \overset{d}{=} (X_{\pi(i),\tau(j)})$
array $(X_{i,j})$	**jointly exchangeable**	$(X_{i,j}) \overset{d}{=} (X_{\pi(i),\pi(j)})$

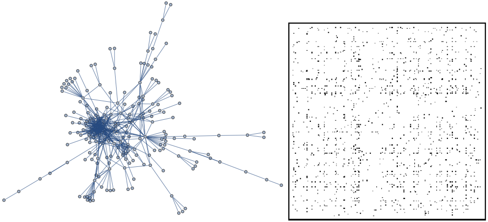
Exchangeable arrays in models of graph/relational data

Definition.

structure	*symmetry*	*definition*
sequence (Y_n)	**exchangeable**	$(Y_n) \overset{d}{=} (Y_{\pi(n)})$
array $(X_{i,j})$	**separately exchangeable**	$(X_{i,j}) \overset{d}{=} (X_{\pi(i),\tau(j)})$
array $(X_{i,j})$	**jointly exchangeable**	$(X_{i,j}) \overset{d}{=} (X_{\pi(i),\pi(j)})$

Example. Adjacency matrix $(X_{i,j})_{i,j\in\mathbb{N}}$ of an undirected graph on $\mathbb{N}$.





Orbanz and **R.** (2014).

Exchangeable arrays in models of graph/relational data

Theorem (Aldous-Hoover). $\theta, U_i, V_j, W_{i,j}$ *all i.i.d. uniform.*

structure	symmetry	representation

Exchangeable arrays in models of graph/relational data

Theorem (Aldous-Hoover). $\theta, U_i, V_j, W_{i,j}$ *all i.i.d. uniform.*

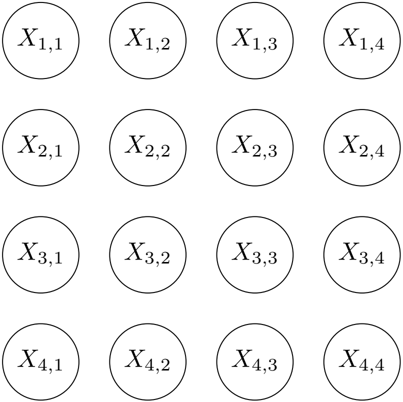
structure	symmetry	representation
array $(X_{i,j})$	$(X_{i,j}) \overset{d}{=} (X_{\pi(i),\tau(j)})$	$(X_{i,j}) \overset{d}{=} (f(\theta, V_i, U_j, W_{i,j}))$

Exchangeable arrays in models of graph/relational data

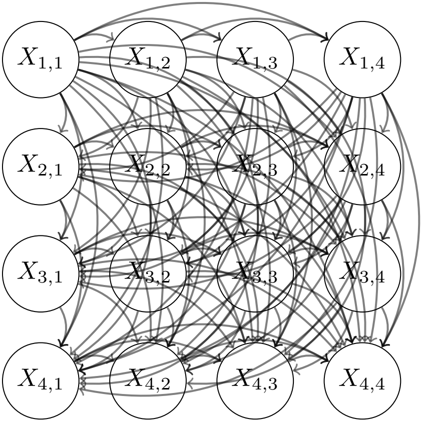
Theorem (Aldous-Hoover). $\theta, U_i, V_j, W_{i,j}$ *all i.i.d. uniform.*

structure	symmetry	representation
array $(X_{i,j})$	$(X_{i,j}) \overset{d}{=} (X_{\pi(i),\tau(j)})$	$(X_{i,j}) \overset{d}{=} (f(\theta, V_i, U_j, W_{i,j}))$
sequence (Y_n)	$(Y_n) \overset{d}{=} (Y_{\pi(n)})$	$(Y_n) \overset{d}{=} (f(\theta, U_n))$

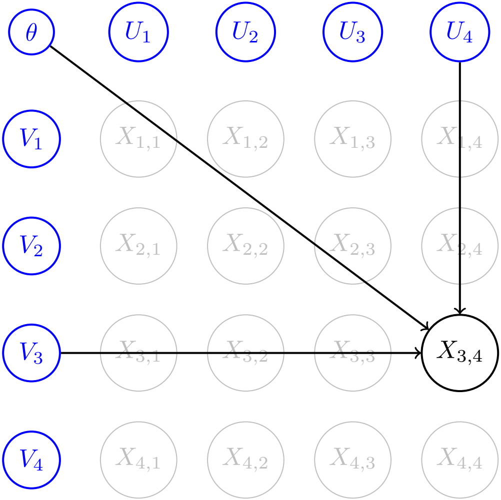
Visualization of Aldous-Hoover theorem for exchangeable arrays



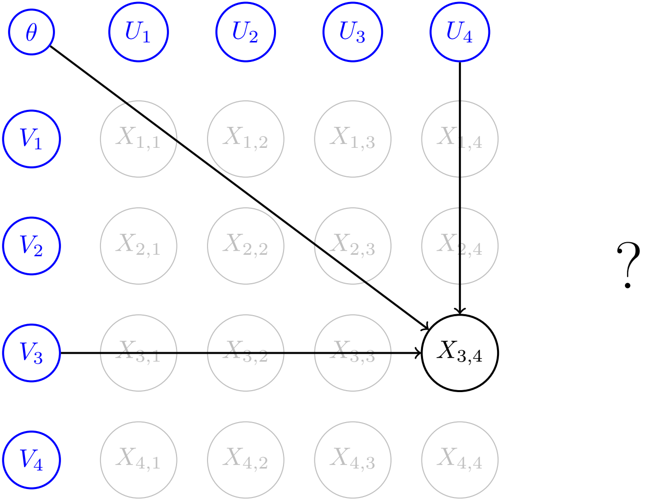
Visualization of Aldous-Hoover theorem for exchangeable arrays



Visualization of Aldous-Hoover theorem for exchangeable arrays



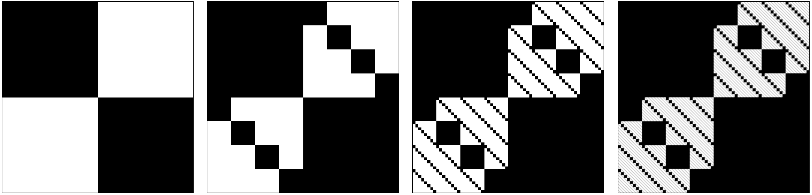
Visualization of Aldous-Hoover theorem for exchangeable arrays



Q: Is the Aldous-Hoover theorem computable?
A: **No.**

Theorem (Avigad, Freer, R., and Rute). *There is an exchangeable array X with a computable distribution but no a.e. computable f satisfying Aldous-Hoover.*

Even "computationally universal" probabilistic programming languages cannot represent certain conditional independence structure.



Computably-distributed array X, noncomputable f [AF**R**R]

The construction (an aliens dating site).

- Let rows/columns represent aliens.
  $X_{i,j} = 1$ means aliens i and j are matched.
- Each alien answers an infinitely-long questionnaire.
- Question $k \in \{1, 2, \dots\}$ has 2^k possible answers.
- Aliens hate answering questionnaires, so they answer randomly.
- Two aliens are matched if they agree on ANY question.

Computably-distributed array X, noncomputable f [AF**R**R]

The construction (an aliens dating site).

- Let rows/columns represent aliens.
  $X_{i,j} = 1$ means aliens i and j are matched.
- Each alien answers an infinitely-long questionnaire.
- Question $k \in \{1, 2, \dots\}$ has 2^k possible answers.
- Aliens hate answering questionnaires, so they answer randomly.
- Two aliens are matched if they agree on ANY question.

Proof sketch.

- Note: f is "return 1 iff two aliens agree somewhere".
- **(f not a.e. computable)** [Topological obstruction.]
  Given two questionnaires, can't accurately check in finite time.
- **(array computably-distributed)**
  The probability of agreeing on any question $n, n+1, \dots$ decays.
  Using only first n questions yields an approximation.

Approximating f sufficed to sample. **Q: The converse?**

Silver lining? f is always "nearly computable"

Let μ be a computable probability measure.

Definition. Say f is **a.e. computable** when
we can compute f on a set of μ-measure one.

Definition (Kriesel-Lacombe (1957), Šanin (1968), Ko (1986)).
Say f is **computably measurable** when,
uniformly for any $\varepsilon > 0$,
we can compute f on a set of μ-measure at least $1 - \varepsilon$.

Theorem (Avigad, Freer, R., and Rute). *The distribution of an exchangeable array X is computable if and only if there is a computably measurable function f satisfying Aldous-Hoover.*

Exchangeability and probabilistic programming

Exchangeable random structures possess a lot of structure.

$$(Y_i) \stackrel{d}{=} (f(\theta, U_i))$$
$$(X_{i,j}) \stackrel{d}{=} (f(\theta, U_i, V_j, W_{i,j}))$$

Can your favorite PPL represent f?

Exchangeability and probabilistic programming

Exchangeable random structures possess a lot of structure.

$$(Y_i) \overset{d}{=} (f(\theta, U_i))$$
$$(X_{i,j}) \overset{d}{=} (f(\theta, U_i, V_j, W_{i,j}))$$

Can your favorite PPL represent f?

Theorem (FR12). *f a.e. computable for sequences.*
Theorem (AFRR). *f merely computably measurable for arrays.*

Approximation essential for capturing structure.

Exchangeability and probabilistic programming

Exchangeable random structures possess a lot of structure.

$$(Y_i) \overset{d}{=} (f(\theta, U_i))$$
$$(X_{i,j}) \overset{d}{=} (f(\theta, U_i, V_j, W_{i,j}))$$

Can your favorite PPL represent f?

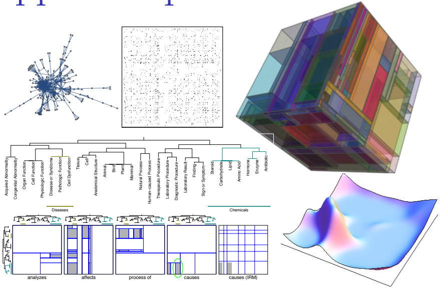
Theorem (FR12). *f a.e. computable for sequences.*
Theorem (AFRR). *f merely computably measurable for arrays.*

Approximation essential for capturing structure.

But do such arrays appear in practice?

Do such arrays X appear in practice?

YES!



√ a.e. computable f

× merely computably measurable f

√ **Infinite Relational Model**

(Kemp, Tenenbaum, Griffiths, Yamada, and Ueda 2008)

√ **Linear Relational Model**

(**R.** and Teh 2009)

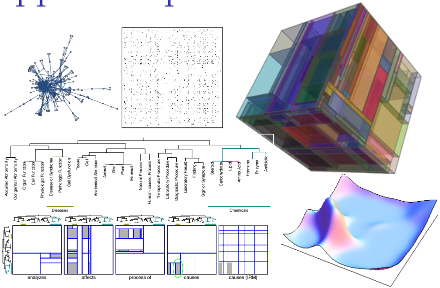
× **Infinite Feature Relational Model**

(Miller, Griffiths, and Jordan 2010)

× **Random Function Model**

(Lloyd, Orbanz, **R.**, and Ghahramani 2012)

Do such arrays X appear in practice?



YES!

√ a.e. computable f
× merely computably measurable f

√ **Infinite Relational Model**

(Kemp, Tenenbaum, Griffiths, Yamada, and Ueda 2008)

Dirichlet process

√ **Linear Relational Model**

(**R.** and Teh 2009)

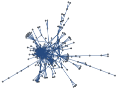
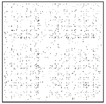
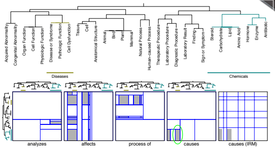
× **Infinite Feature Relational Model**

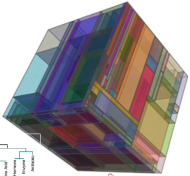
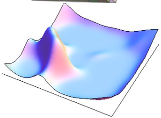
(Miller, Griffiths, and Jordan 2010)

× **Random Function Model**

(Lloyd, Orbanz, **R.**, and Ghahramani 2012)

Do such arrays X appear in practice?

YES!

√ a.e. computable f

× merely computably measurable f

√ **Infinite Relational Model** — Dirichlet process

(Kemp, Tenenbaum, Griffiths, Yamada, and Ueda 2008)

√ **Linear Relational Model** — Mondrian process

(**R.** and Teh 2009)

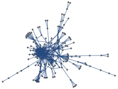
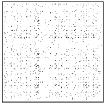
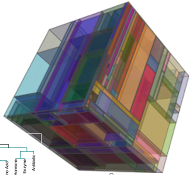
× **Infinite Feature Relational Model**

(Miller, Griffiths, and Jordan 2010)

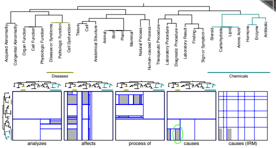
× **Random Function Model**

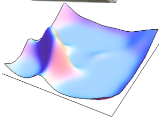
(Lloyd, Orbanz, **R.**, and Ghahramani 2012)

Do such arrays X appear in practice?

YES!





√ a.e. computable f
× merely computably measurable f

√ **Infinite Relational Model** Dirichlet process
(Kemp, Tenenbaum, Griffiths, Yamada, and Ueda 2008)

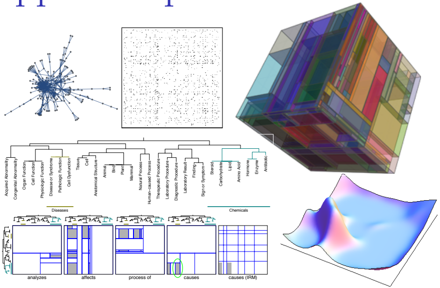
√ **Linear Relational Model** Mondrian process
(**R.** and Teh 2009)

× **Infinite Feature Relational Model** Beta process
(Miller, Griffiths, and Jordan 2010)

× **Random Function Model**
(Lloyd, Orbanz, **R.**, and Ghahramani 2012)

Do such arrays X appear in practice?

YES!



√ a.e. computable f
× merely computably measurable f

√ **Infinite Relational Model** — Dirichlet process
(Kemp, Tenenbaum, Griffiths, Yamada, and Ueda 2008)

√ **Linear Relational Model** — Mondrian process
(**R.** and Teh 2009)

× **Infinite Feature Relational Model** — Beta process
(Miller, Griffiths, and Jordan 2010)

× **Random Function Model** — Gaussian process
(Lloyd, Orbanz, **R.**, and Ghahramani 2012)

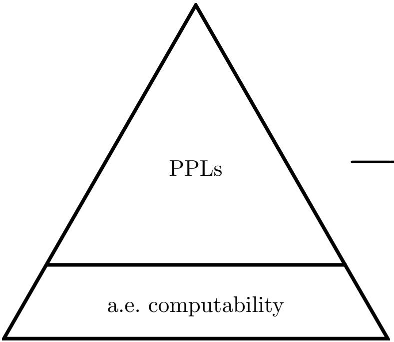
Talk Outline

1. The stochastic inference problem
2. Where are we now in probabilistic programming?
3. Approximability and Exchangeability:
   When can we represent conditional independence?
4. Conclusion

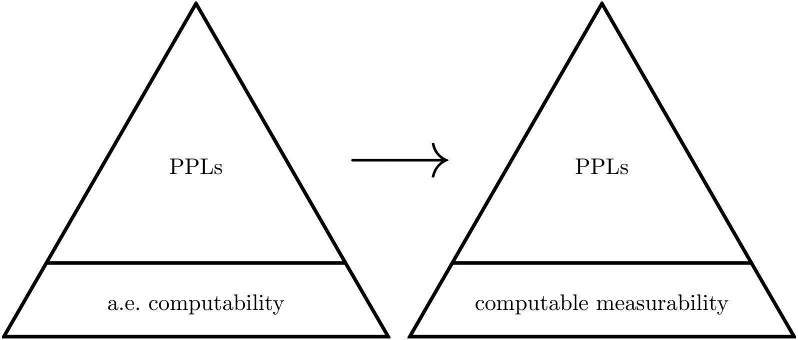
Talk Outline

1. The stochastic inference problem
2. Where are we now in probabilistic programming?
3. Approximability and Exchangeability:
   When can we represent conditional independence?
4. **Conclusion**

Conclusion

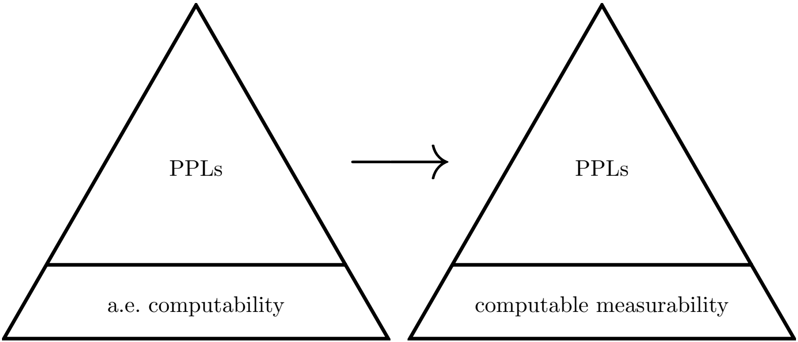


Conclusion



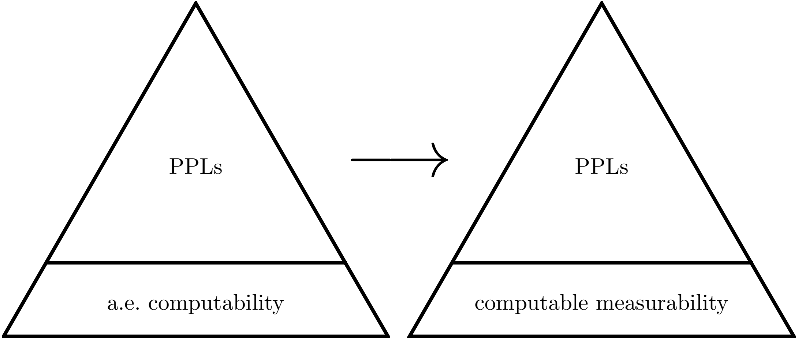
1. **One can see the gap in the literature.**
   Key stochastic processes are merely computably measurable.

Conclusion



1. **One can see the gap in the literature.**
   Key stochastic processes are merely computably measurable.
2. **How do we use such representations?**
   Exact-approximate inference and computable measurability?

Conclusion



1. **One can see the gap in the literature.**
   Key stochastic processes are merely computably measurable.
2. **How do we use such representations?**
   Exact-approximate inference and computable measurability?
3. **Need new programming language constructs.**
   Naïvely, we would need to thread ε's everywhere in program.