ngoodman@
stanford.edu

Languages for probability

Noah D. Goodman
Stanford University

IPAM graduate summer school
July 6, 2011

Motivation

How should we represent knowledge?

That is useful for drawing conclusions.

In an uncertain world.

Motivation

That is useful for drawing conclusions.

In an uncertain world.

Motivation

In an uncertain world.

Motivation

Logic

- Fix a set of *atomic propositions* (variables).
- A truth-table describes *possible* worlds.

cough	sneeze	flu	TB	Possible?
t	t	t	t	y
t	t	f	f	n
...	...	...	...	...
f	f	f	f	y

Logic

- Do inference by crossing out rows.
- What is true in remaining worlds?
- Problems:
  - Size of truth table is exponential.
  - Inference becomes impractical.

cough	flu	TB	Possible?
t	t	t	y
t	t	f	y
t	f	t	y
t	f	f	n
f	t	t	n
f	t	f	n
f	f	t	n
f	f	f	y

Logic

- Do inference by crossing out rows.
- What is true in remaining worlds?
- Problems:
  - Size of truth table is exponential.
  - Inference becomes impractical.

If cough=t, flu=f, TB?

cough	flu	TB	Possible?
t	t	t	y
t	t	f	y
t	f	t	y
t	f	f	n
f	t	t	n
f	t	f	n
f	f	t	n
f	f	f	y

Logic

- Do inference by crossing out rows.
- What is true in remaining worlds?
- Problems:
  - Size of truth table is exponential.
  - Inference becomes impractical.

If cough=t, flu=f, TB?

cough	flu	TB	Possible?
t	t	t	y
t	t	f	y
t	f	t	y
t	f	f	n
f	t	t	n
f	t	f	n
f	f	t	n
f	f	f	y

Logic

- Do inference by crossing out rows.
- What is true in remaining worlds?
- Problems:
  - Size of truth table is exponential.
  - Inference becomes impractical.

If cough=t, flu=f, TB?

cough	flu	TB	Possible?
t	t	t	y
t	t	f	y
t	f	t	y
t	f	f	n
f	t	t	n
f	t	f	n
f	f	t	n
f	f	f	y

Logic

- Do inference by crossing out rows.
- What is true in remaining worlds?
- Problems:
  - Size of truth table is exponential.
  - Inference becomes impractical.

If cough=t, flu=f, TB?

cough	flu	TB	Possible?
t	t	t	y
t	t	f	y
t	f	t	y
t	f	f	n
f	t	t	n
f	t	f	n
f	f	t	n
f	f	f	y

Logic

- Do inference by crossing out rows.
- What is true in remaining worlds?
- Problems:
  - Size of truth table is exponential.
  - Inference becomes impractical.

If cough=t, flu=f, TB? TB=t.

cough	flu	TB	Possible?
t	t	t	y
t	t	f	y
t	f	t	y
t	f	f	n
f	t	t	n
f	t	f	n
f	f	t	n
f	f	f	y

Propositional logic

- Solution: use a formal language to describe possible worlds more compactly.
- Operators defined by conditional truth tables:

A	B	(A *or* B)
t	t	t
t	f	t
f	t	t
f	f	f

- Describe possible worlds by *composing* operators.

Example

cough	flu	TB	Possible?
t	t	t	y
t	t	f	y
t	f	t	y
t	f	f	y
f	t	t	n
f	t	f	n
f	f	t	n
f	f	f	y

= (cough *if* (flu *or* TB)

Propositional logic

- Propositional language:
  - Key idea: compositionality of CTTs.
  - Representation is (sometimes) compact.
  - New inference methods (proof rules).
  - More intuitive specification.
  - Mental representation?
    (See: G. Boole, 1854, “The laws of thought”.)

Languages for logic

- Purposes of a language:
  - Makes writing down knowledge easier.
  - Makes reasoning about knowledge clearer.
  - Supports efficient inference.
  - Gives ideas about mental representation.

Higher logics

- Propositional logic is not compositional *enough:*
  - must fix the set of variables in advance,
  - misses much compactness.
- Key idea: compositionality can still work if not all the pieces are truth functions by themselves.
  - Boolean operators: from truth to truth.
  - Predicates: from objects to truth.
  - Higher-order functions...

Predicate logic

- First order logic: $\forall x \exists y (R(x, y) \wedge S(x))$
  - Predicates are functions from objects to truth values.
  - Quantifiers assign object variables to objects.
- FoL can compress much more...
  - Each predicate+objects could be an atomic proposition.
  - FoL rules apply to many such combinations.
  - Rules of chess: 1 page in FoL, 1000 in propositional (according to Stuart Russell).

Functional logic

- By allowing functions of higher type (such as functions on functions) we can compress yet more.
- λ-calculus: only function creation and application, but it is computationally universal!
- More later....

Probability

- Instead of *possibility* of worlds represent *degree of belief* in each world -- a distribution.
  - A number on each possible world
  - Sum over all worlds is 1.

cough	sneeze	flu	TB	Prob
t	t	t	t	0.1
t	t	f	f	0.02
...		...	...	...
f	f	f	f	0.3

Computing conditionals

cough	sneeze	flu	TB	Prob
t	t	t	t	0.1
t	t	f	f	0.02
...		...	...	...
f	f	f	f	0.3

Computing conditionals

cough	sneeze	flu	TB	Prob
t	t	t	t	0.1
t	t	f	f	0.02
...		...	...	...
f	f	f	f	0.3

P(flu | cough)=?

Computing conditionals

cough	sneeze	flu	TB	Prob
t	t	t	t	0.1
t	t	f	f	0.02
...		...	...	...
f	f	f	f	0.3

P(flu | cough)=?

Cross out rows where $B \neq b$ and renormalize.

Computing conditionals

cough	sneeze	flu	TB	Prob
t	t	t	t	0.1
t	t	f	f	0.02
...		...	...	...
f	f	f	f	0.3

P(flu | cough)=?

Cross out rows where $B \neq b$ and renormalize.

$$P(A = a|B = b) = \frac{P(A = a, B = b)}{P(B = b)}$$

Computing conditionals

cough	sneeze	flu	TB	Prob
t	t	t	t	0.1
t	t	f	f	0.02
...		...	...	...
f	f	f	f	0.3

P(flu | cough)=?

Cross out rows where $B \neq b$ and renormalize.

$$P(A = a|B = b) = \frac{P(A = a, B = b)}{P(B = b)}$$

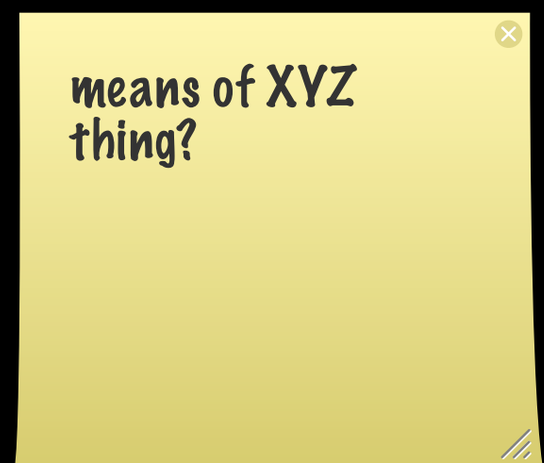
$$P(B = b) = \sum_{a'} P(A = a', B = b)$$

Probability

- Probability is (arguably) the right way to reason under uncertainty.
- Many appealing effects: non-monotonic reasoning, occam's razor, learning-to-learn, etc.
- Problems? Same as for logic: the table grows exponentially in the number of variables.
- Compositional languages for probability?

Languages for probability

- Purposes of a language:
  - Makes writing down models easier.
  - Makes reasoning about models clearer.
  - Supports efficient inference.
  - Gives ideas about mental representation.



Distributions

- We begin with a fixed set of random variables, $x_1, ..., x_n$.
- We want to represent a joint distribution $P(x_1, ..., x_n)$ more compactly and intuitively than a table of $2^n - 1$ numbers.

cough	sneeze	flu	TB	Prob
t	t	t	t	0.1
t	t	f	f	0.02
...		...	...	...
f	f	f	f	0.3

Factoring

- Recall: $P(A|B) = \frac{P(A,B)}{P(B)}$
- Thus we can factor the joint distribution into a product of conditional distributions:

$$P(x_1, ..., x_n) = P(x_1)P(x_2|x_1)...P(x_n|x_1, ..., x_{n-1})$$

- This is different but not more compact.

TB = t
0.1

TB	flu = t
t	0.2
f	0.2

TB	flu	cough = t
t	t	0.9
t	f	0.8
f	t	0.75
f	f	0.1

TB	flu	cough	sneeze = t
t	t	t	0.8
t	t	f	0.8
...		...	...
f	f	f	0.2

Independence

- Variable x_2 is *independent* of x_1 if $P(x_2|x_1) = P(x_2)$ (i.e. same for all values of x_1).
- If the distribution has independence (and we choose the right order for the conditionals) we can get away with fewer numbers...

TB = t
0.1

flu = t
0.2

TB	flu	cough = t
t	t	0.9
t	f	0.8
f	t	0.75
f	f	0.1

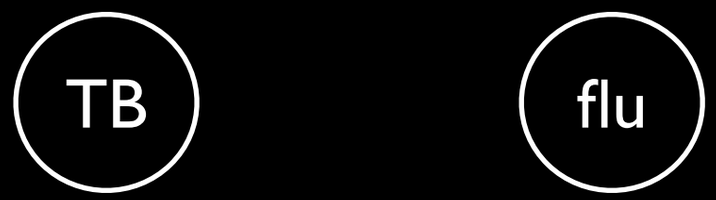
flu	sneeze = t
t	0.8
f	0.2

Bayes nets

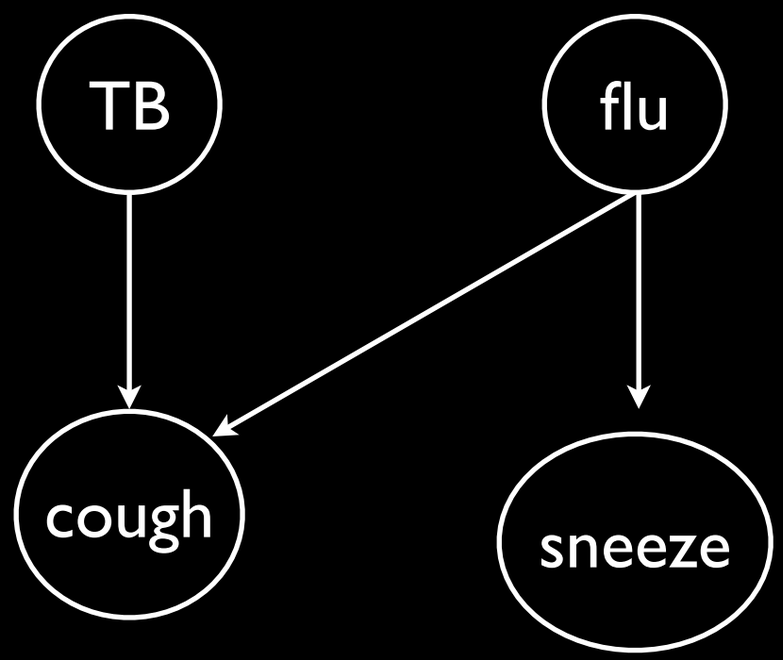
- A way to specify a distribution: give the (in)dependence structure, then the conditional probability tables (CPTs).
- A *Bayes net* or *directed graphical model:*
  - A set of variables V.
  - A directed acyclic graph on V.
  - A CPT for each variable in V given it's parents in the graph.

See Pearl, 1988

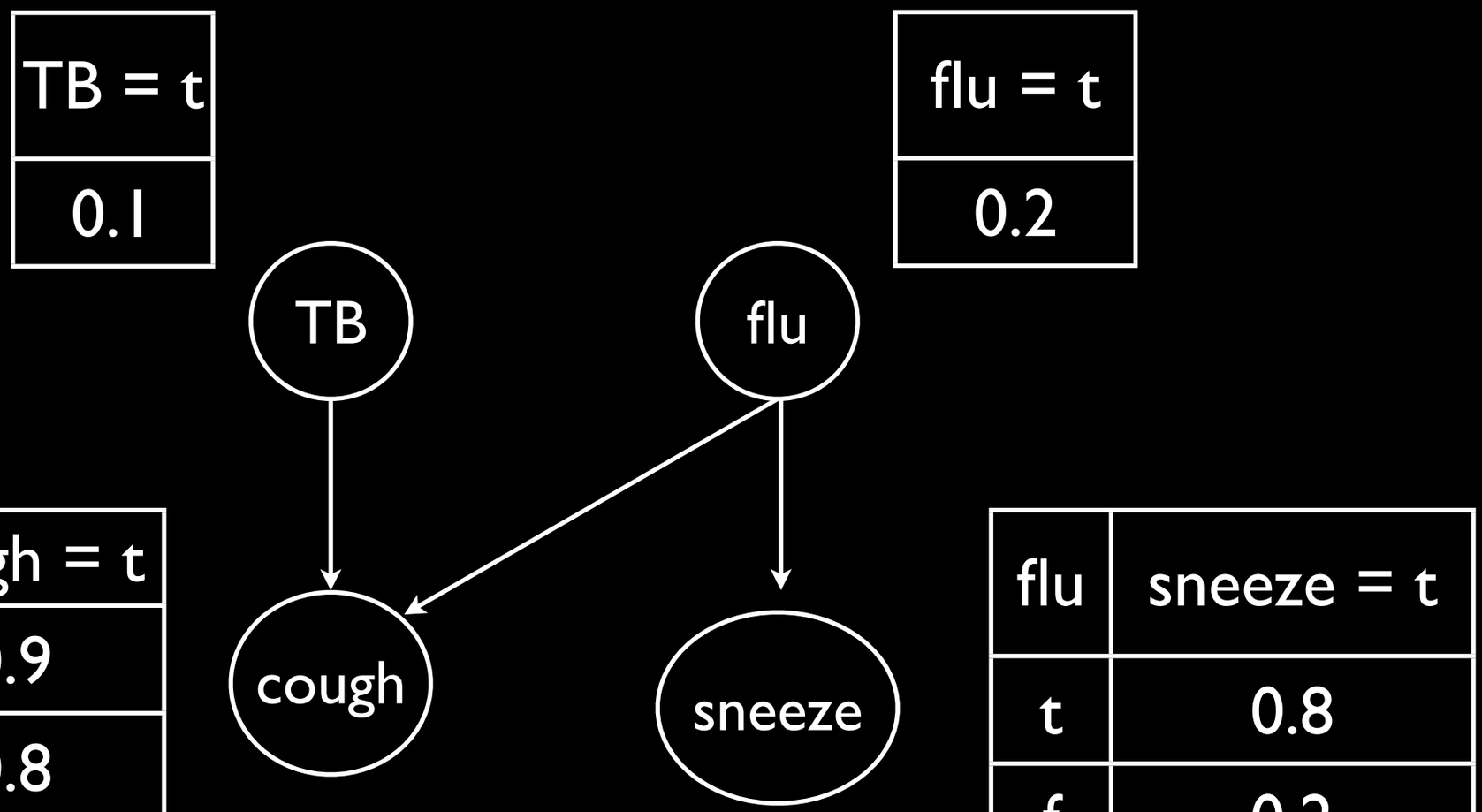
Example



Example



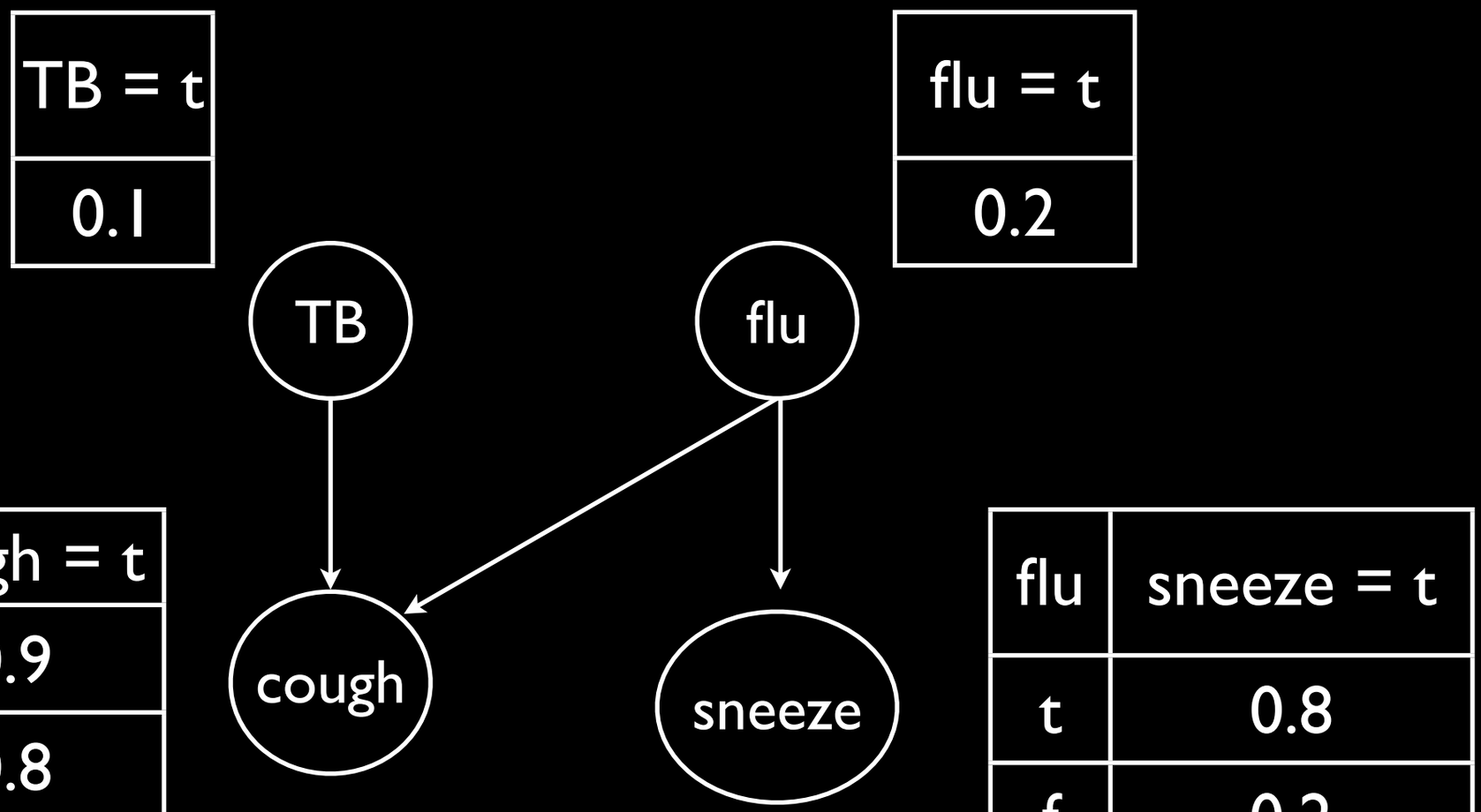
Example



TB	flu	cough = t
t	t	0.9
t	f	0.8
f	t	0.75
f	f	0.1

flu	sneeze = t
t	0.8
f	0.2

Example



TB	flu	cough = t
t	t	0.9
t	f	0.8
f	t	0.75
f	f	0.1

flu	sneeze = t
t	0.8
f	0.2

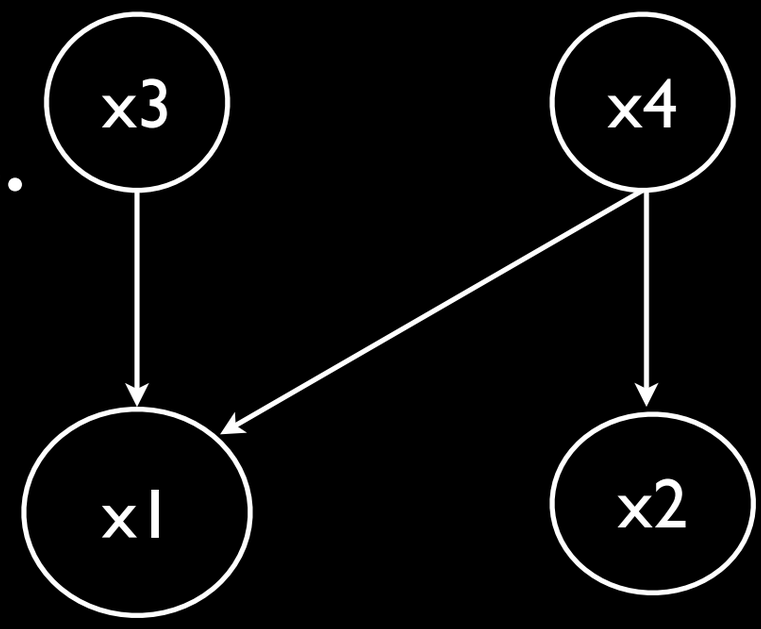
Note: names and graph are redundant (arrows as “binding”).

Aside

- A Bayes net is a *representation* for a distribution.
- Bayesian statistics describes rules for *inference* given a model.
  - Bayes rule is a useful pattern common when doing inference from observation to hypothesized causes.

Computing conditionals

- Using the independence structure can simplify computing conditionals.



$$P(x_2, x_3, x_4 | x_1{=}1) = \frac{P(x_2, x_3, x_4, x_1{=}1)}{P(x_1{=}1)}$$

$$P(x_1{=}1) = \sum_{x_2, x_3, x_4} P(x_2, x_3, x_4, x_1{=}1)$$

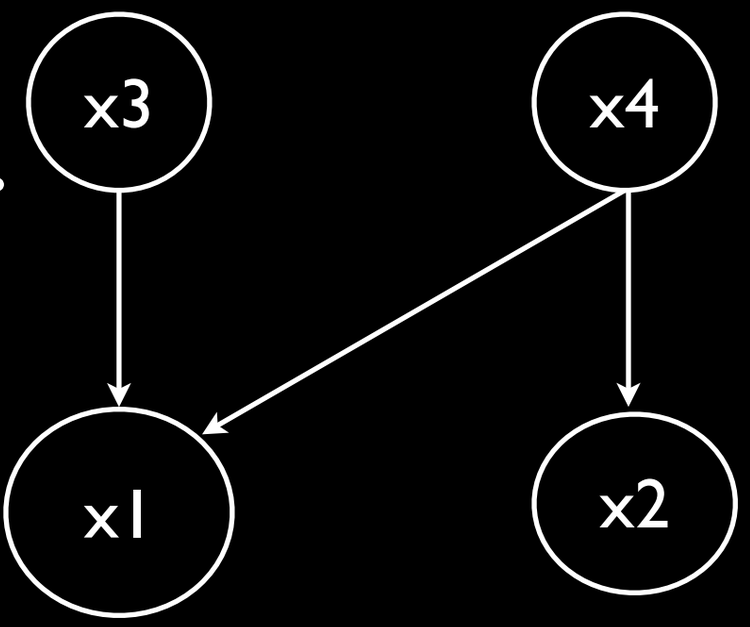
$$= \sum_{x_2, x_3, x_4} P(x_3)P(x_4)P(x_1{=}1 | x_3, x_4)P(x_2 | x_4)$$

$$= \sum_{x_2, x_3, x_4} P(x_3)P(x_4)P(x_1{=}1 | x_3, x_4)$$

$$= \sum_{x_3} P(x_3) \sum_{x_4} P(x_4)P(x_1{=}1 | x_3, x_4)$$

Computing conditionals

- Using the independence structure can simplify computing conditionals.



$$P(x_2, x_3, x_4 | x_1{=}1) = \frac{P(x_2, x_3, x_4, x_1{=}1)}{P(x_1{=}1)}$$

$$P(x_1{=}1) = \sum_{x_2, x_3, x_4} P(x_2, x_3, x_4, x_1{=}1)$$

sum over 8 values

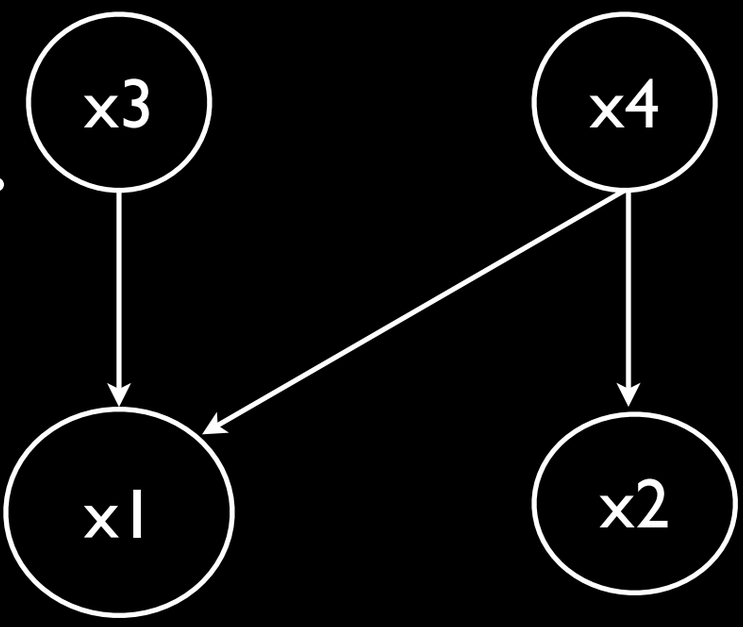
$$= \sum_{x_2, x_3, x_4} P(x_3)P(x_4)P(x_1{=}1|x_3, x_4)P(x_2|x_4)$$

$$= \sum_{x_2, x_3, x_4} P(x_3)P(x_4)P(x_1{=}1|x_3, x_4)$$

$$= \sum_{x_3} P(x_3) \sum_{x_4} P(x_4)P(x_1{=}1|x_3, x_4)$$

Computing conditionals

- Using the independence structure can simplify computing conditionals.



$$P(x_2, x_3, x_4 | x_1{=}1) = \frac{P(x_2, x_3, x_4, x_1{=}1)}{P(x_1{=}1)}$$

$$P(x_1{=}1) = \sum_{x_2, x_3, x_4} P(x_2, x_3, x_4, x_1{=}1)$$

sum over 8 values

$$= \sum_{x_2, x_3, x_4} P(x_3)P(x_4)P(x_1{=}1|x_3, x_4)P(x_2|x_4)$$

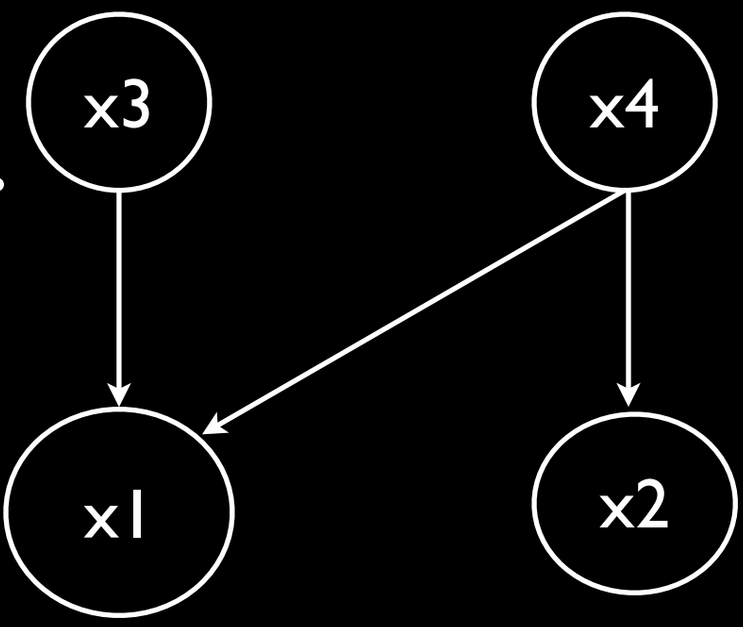
$$= \sum_{x_2, x_3, x_4} P(x_3)P(x_4)P(x_1{=}1|x_3, x_4)$$

sum over 4 values

$$= \sum_{x_3} P(x_3) \sum_{x_4} P(x_4)P(x_1{=}1|x_3, x_4)$$

Computing conditionals

- Using the independence structure can simplify computing conditionals.



$$P(x_2, x_3, x_4 | x_1{=}1) = \frac{P(x_2, x_3, x_4, x_1{=}1)}{P(x_1{=}1)}$$

$$P(x_1{=}1) = \sum_{x_2, x_3, x_4} P(x_2, x_3, x_4, x_1{=}1)$$

sum over 8 values

$$= \sum_{x_2, x_3, x_4} P(x_3)P(x_4)P(x_1{=}1|x_3, x_4)P(x_2|x_4)$$

$$= \sum_{x_2, x_3, x_4} P(x_3)P(x_4)P(x_1{=}1|x_3, x_4)$$

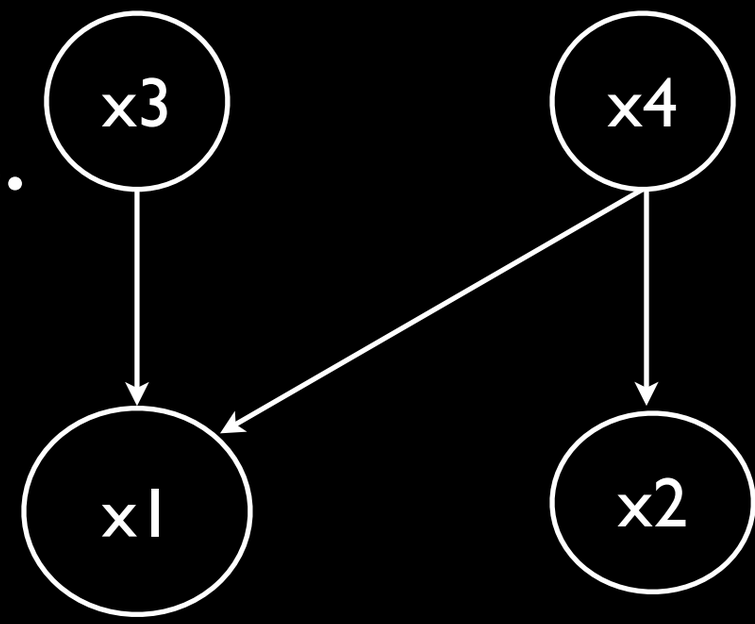
sum over 4 values

$$= \sum_{x_3} P(x_3) \sum_{x_4} P(x_4)P(x_1{=}1|x_3, x_4)$$

sum over 2 then 2 values

Computing conditionals

- Using the independence structure can simplify computing conditionals.
  - Sum/product variable elimination algorithm.
  - Dynamic programming. (Stuhlmueller)
  - Message passing.
  - Variational. (Salakhutdinov)
  - Monte Carlo. (Murray)

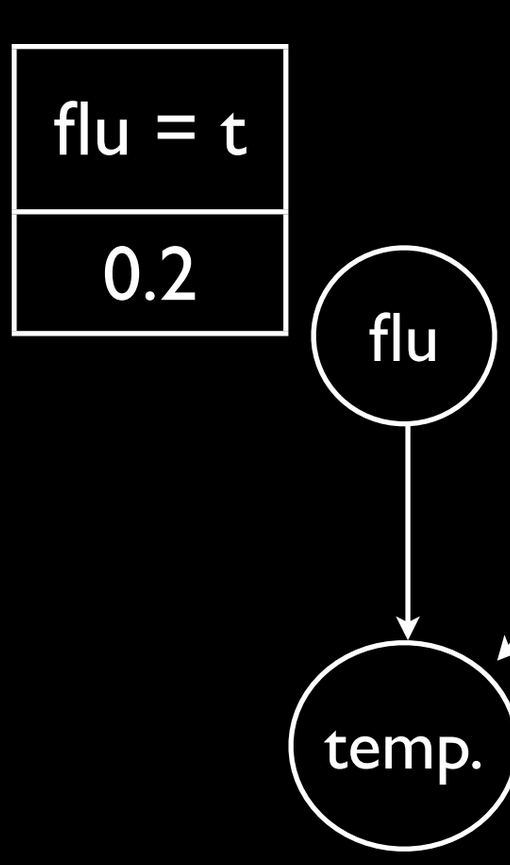


bigger examples =
bigger gains

Non-boolean variables

- Nothing changes when using variables with many values...
  - except CPTs become harder (or impossible) to write!
- Statistician's notation is often used instead: $x \sim D(Pa(x))$, which means x is a sample from the distribution $D(Pa(x))$.

Example



$$\text{diet} \sim Gauss(10.0, 1.0)$$

$$\text{weight} \sim Gauss(\text{diet}, 10.0)$$

$$\text{temp} \sim Gauss(\delta_{\text{flu}}, \text{weight})$$

$$Gauss(\mu, \nu) \propto e^{-\frac{(x-\mu)^2}{2\nu}}$$

Non-boolean variables

- In $x \sim D(Pa(x))$, the D is a *stochastic function* mapping parent assignments to distributions.
  - A graphical model describes a composition of stochastic functions.
  - (But what language are the functions in?)

Causality

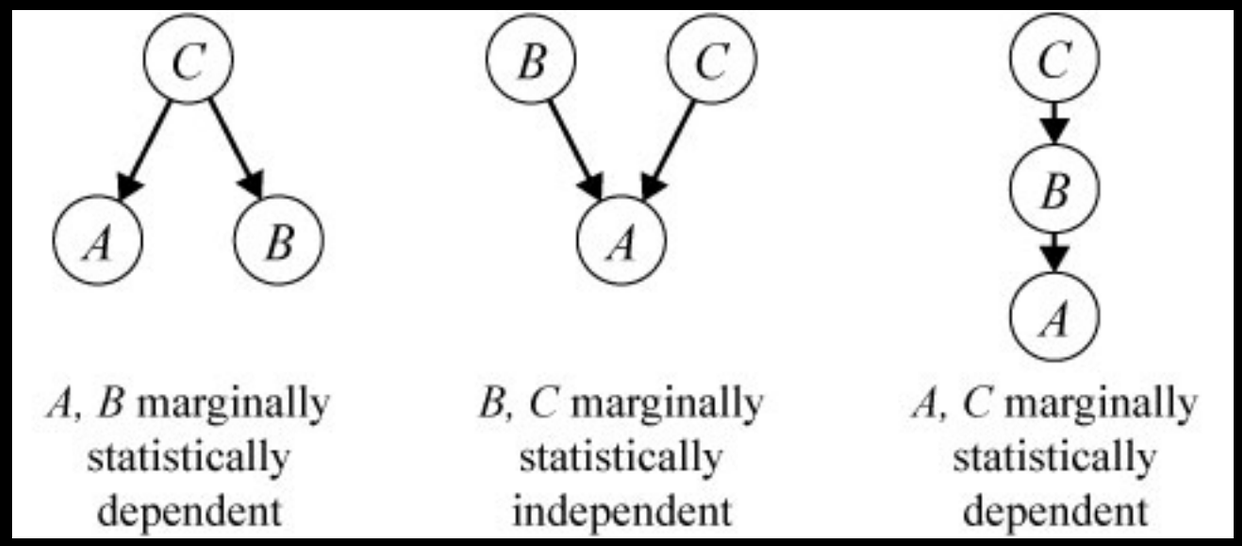
- Directed graphical models also capture a key intuition: probabilistic models describe *causal processes*.
  - The sequence of choices that must be made to construct a world.
  - This is not the flow of time, but of dependence (a partial order).
  - Important for expressing knowledge of how the world works.

Two kinds of dependence

- Causal dependence: A must be sampled in order to sample B.
  (Cf. causality via intervention. Pearl, 2000.)
  - Causal dependence can be read immediately from directed graphical model.
- Statistical dependence: learning something about A provides information about B.
  - Statistical dependence can be reasoned about from the graphical model.

Conditional dependence

- Graphical model structure simplifies reasoning about models -- “where does information flow?”.



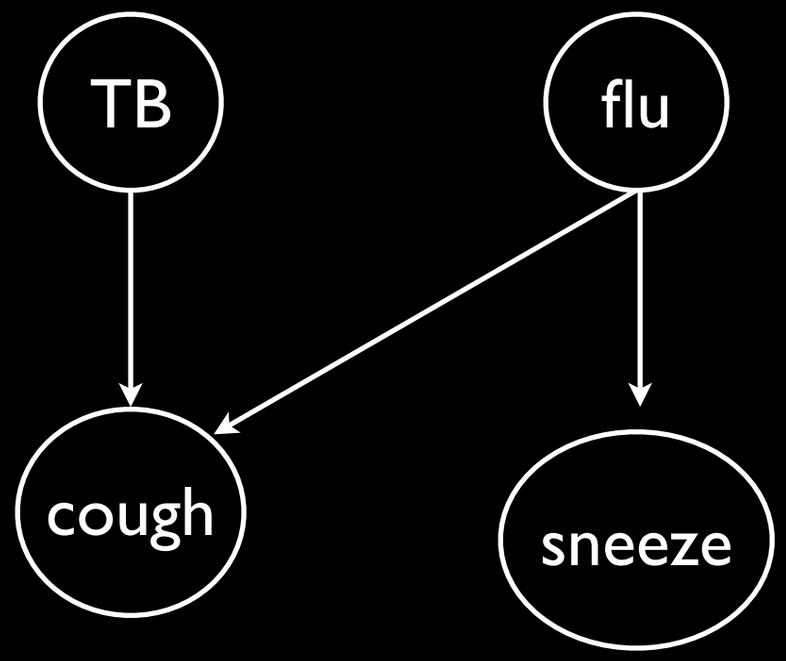
Example: explaining away

TB = t
0.1

flu = t
0.2

flu	sneeze = t
t	0.8
f	0.2

TB	flu	cough = t
t	t	0.9
t	f	0.8
f	t	0.75
f	f	0.1



- TB and flu are marginally (and causally) independent.
- TB and flu conditionally dependent given cough.

Example: explaining away

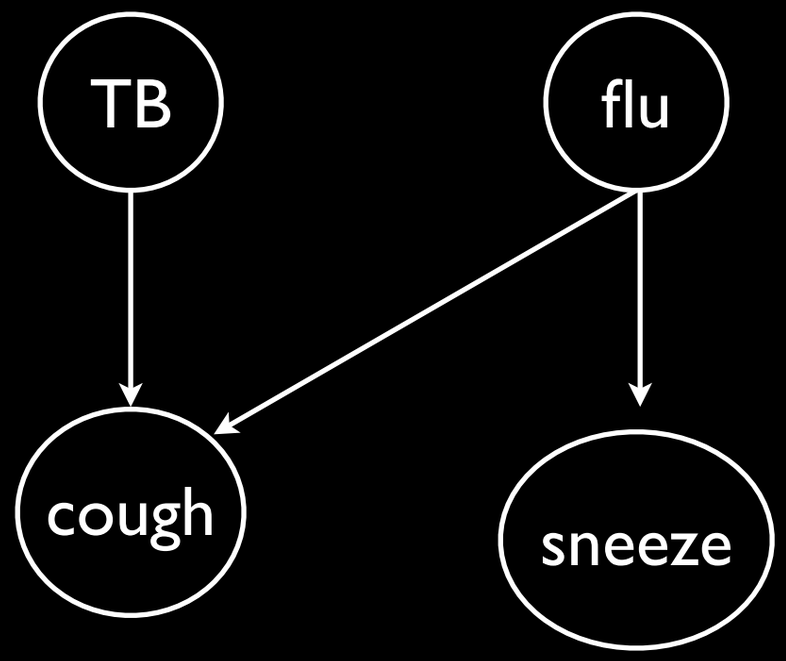
TB = t
0.1

flu = t
0.2

flu	sneeze = t
t	0.8
f	0.2

TB	flu	cough = t
t	t	0.9
t	f	0.8
f	t	0.75
f	f	0.1

$$P(\mathrm{TB}) = 0.1$$



- TB and flu are marginally (and causally) independent.
- TB and flu conditionally dependent given cough.

Example: explaining away

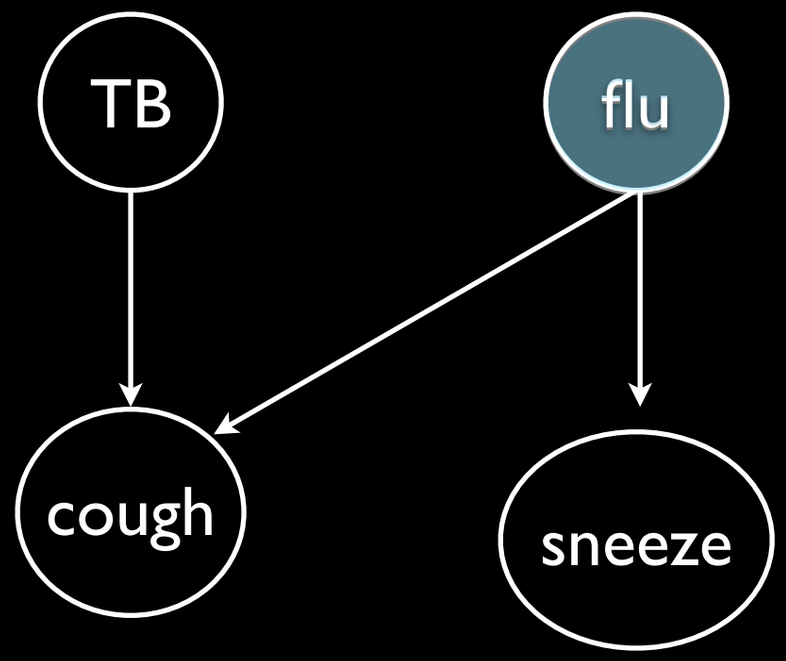
TB = t
0.1

flu = t
0.2

flu	sneeze = t
t	0.8
f	0.2

TB	flu	cough = t
t	t	0.9
t	f	0.8
f	t	0.75
f	f	0.1

$$P(\mathrm{TB}) = 0.1$$



- TB and flu are marginally (and causally) independent.
- TB and flu conditionally dependent given cough.

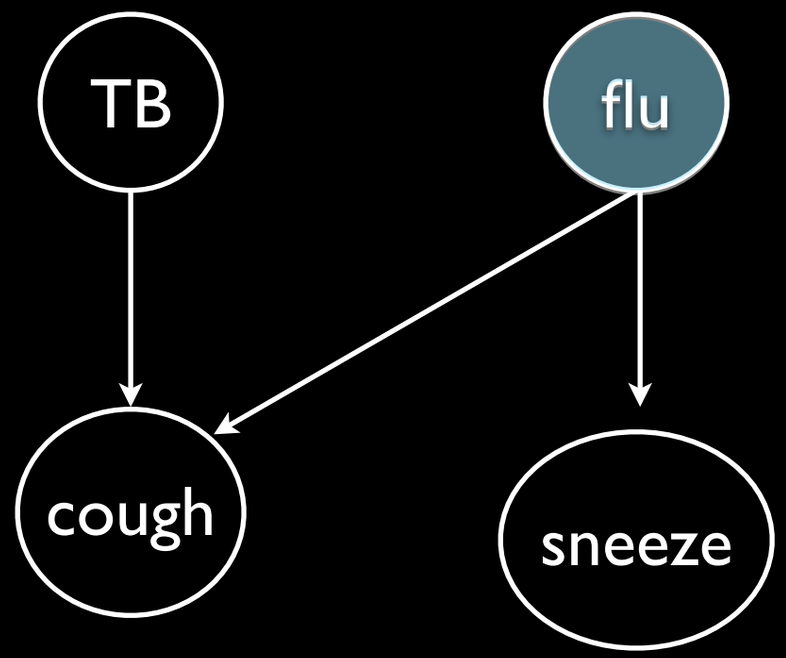
Example: explaining away

TB = t
0.1

flu = t
0.2

flu	sneeze = t
t	0.8
f	0.2

TB	flu	cough = t
t	t	0.9
t	f	0.8
f	t	0.75
f	f	0.1



$$P(\mathrm{TB}) = 0.1$$

$$P(\mathrm{TB}|\mathrm{flu}) = 0.1$$

- TB and flu are marginally (and causally) independent.
- TB and flu conditionally dependent given cough.

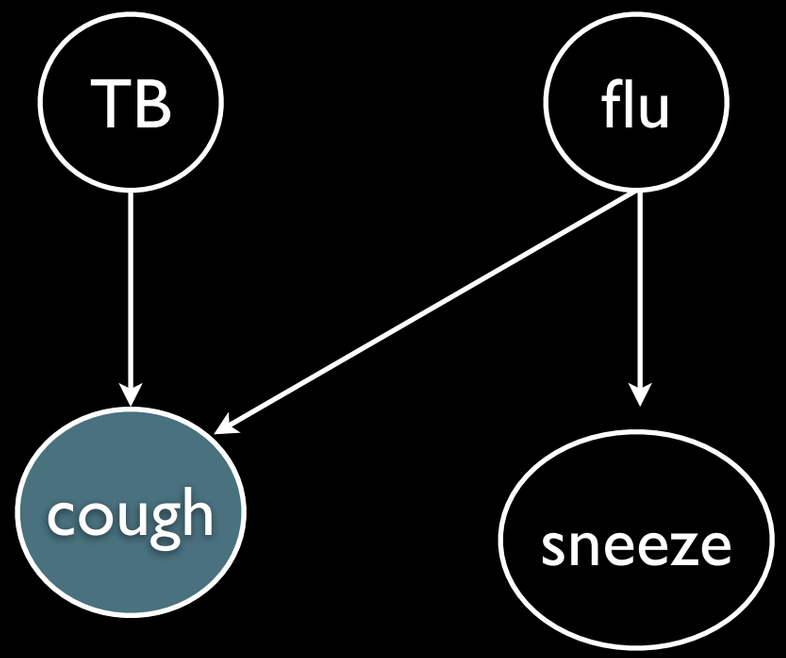
Example: explaining away

TB = t
0.1

flu = t
0.2

flu	sneeze = t
t	0.8
f	0.2

TB	flu	cough = t
t	t	0.9
t	f	0.8
f	t	0.75
f	f	0.1



$$P(\mathrm{TB}) = 0.1$$

$$P(\mathrm{TB}|\mathrm{flu}) = 0.1$$

- TB and flu are marginally (and causally) independent.
- TB and flu conditionally dependent given cough.

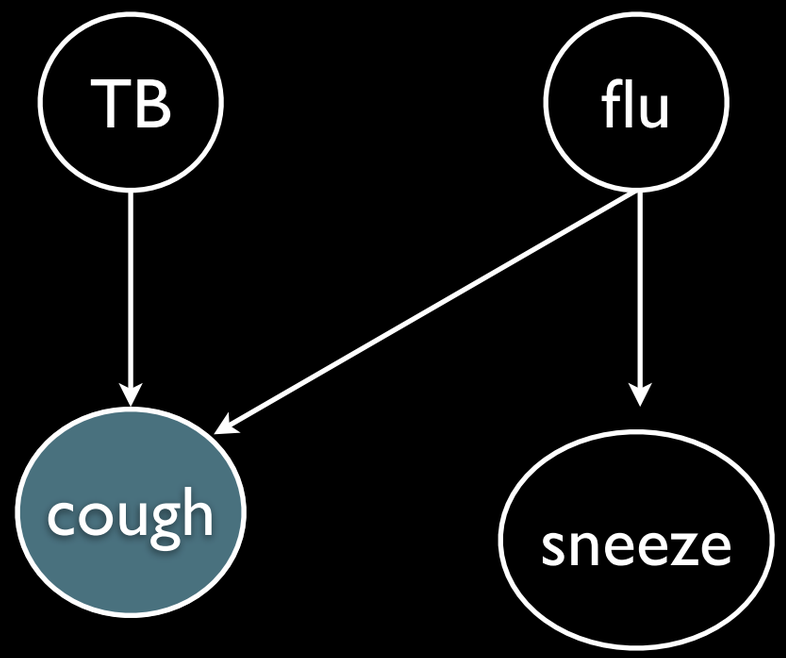
Example: explaining away

TB = t
0.1

flu = t
0.2

flu	sneeze = t
t	0.8
f	0.2

TB	flu	cough = t
t	t	0.9
t	f	0.8
f	t	0.75
f	f	0.1



$$P(\mathrm{TB}) = 0.1$$

$$P(\mathrm{TB}|\mathrm{flu}) = 0.1$$

$$P(\mathrm{TB}|\mathrm{cough}) = 0.293$$

- TB and flu are marginally (and causally) independent.
- TB and flu conditionally dependent given cough.

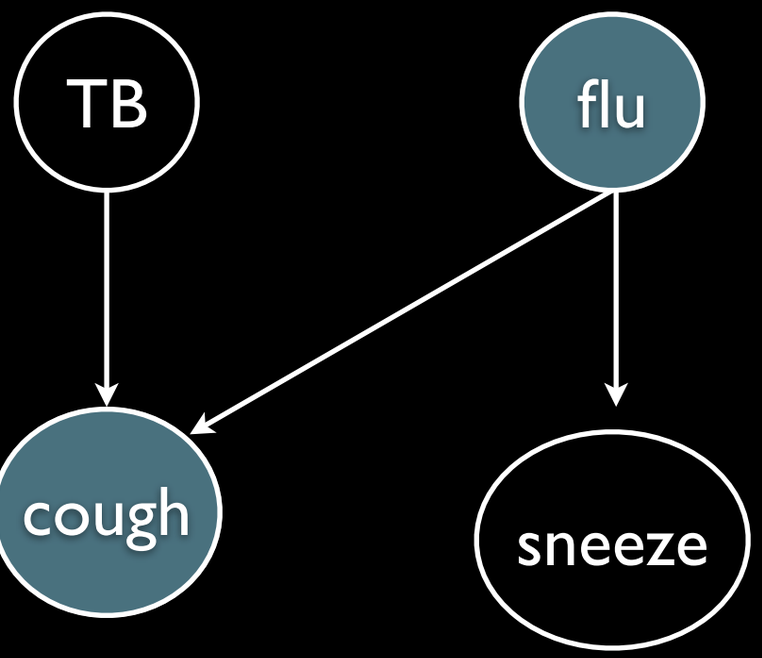
Example: explaining away

TB = t
0.1

flu = t
0.2

flu	sneeze = t
t	0.8
f	0.2

TB	flu	cough = t
t	t	0.9
t	f	0.8
f	t	0.75
f	f	0.1



$$P(\mathrm{TB}) = 0.1$$

$$P(\mathrm{TB}|\mathrm{flu}) = 0.1$$

$$P(\mathrm{TB}|\mathrm{cough}) = 0.293$$

- TB and flu are marginally (and causally) independent.
- TB and flu conditionally dependent given cough.

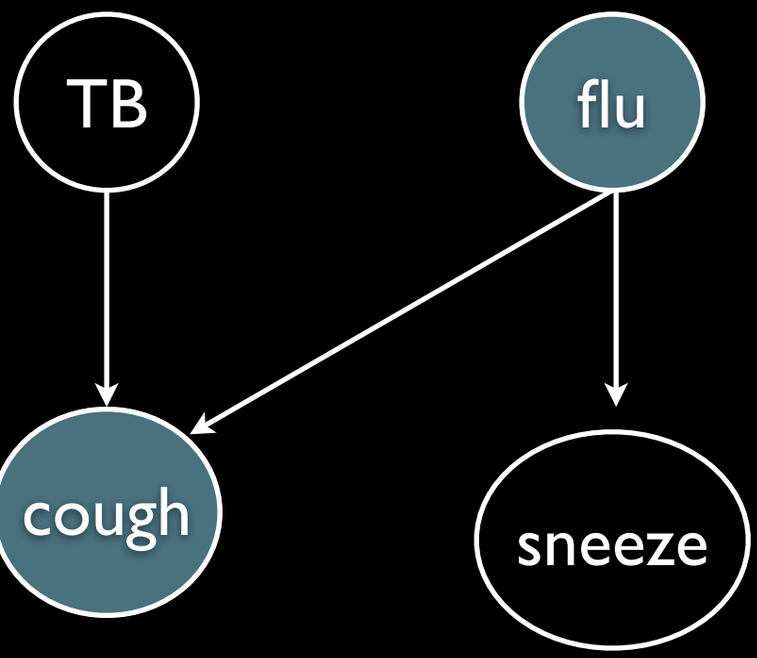
Example: explaining away

TB = t
0.1

flu = t
0.2

flu	sneeze = t
t	0.8
f	0.2

TB	flu	cough = t
t	t	0.9
t	f	0.8
f	t	0.75
f	f	0.1



$$P(\mathrm{TB}) = 0.1$$

$$P(\mathrm{TB}|\mathrm{flu}) = 0.1$$

$$P(\mathrm{TB}|\mathrm{cough}) = 0.293$$

$$P(\mathrm{TB}|\mathrm{cough}, \mathrm{flu}) = 0.128$$

- TB and flu are marginally (and causally) independent.
- TB and flu conditionally dependent given cough.

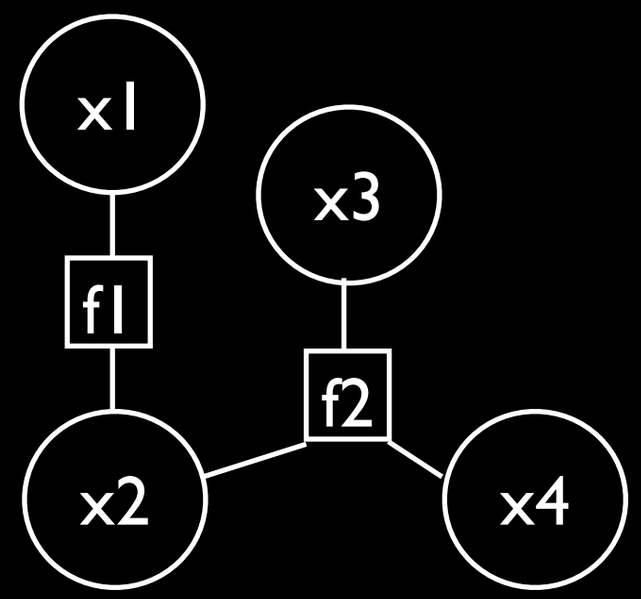
Directed graphical models

- Compact representation for distributions.
- Captures causal process intuition.
- Supports efficient inference.
- Eases reasoning about models.

For more on graphical models see, e.g.,
Koller & Friedman (2009).

Undirected models

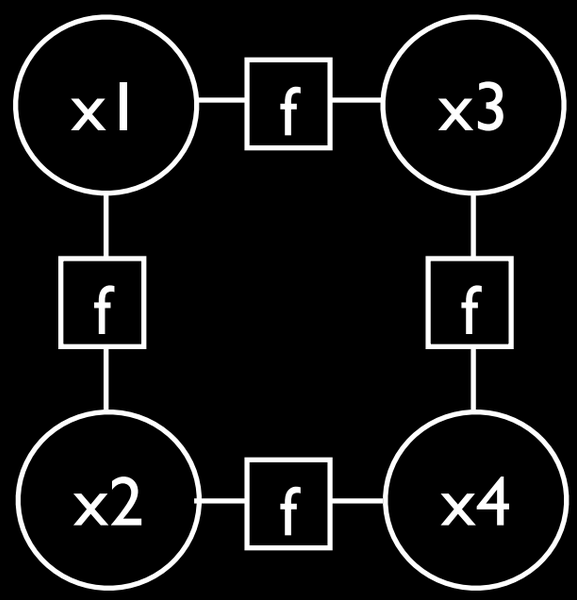
- There is often another way to factor a distribution: write it as an un-normalized product of *factor functions*.
- Factor graph: variable nodes separated by factor nodes, each factor node has a factor function.
- Convenient for expressing non-causal constraints. "Bob goes iff Alice does."



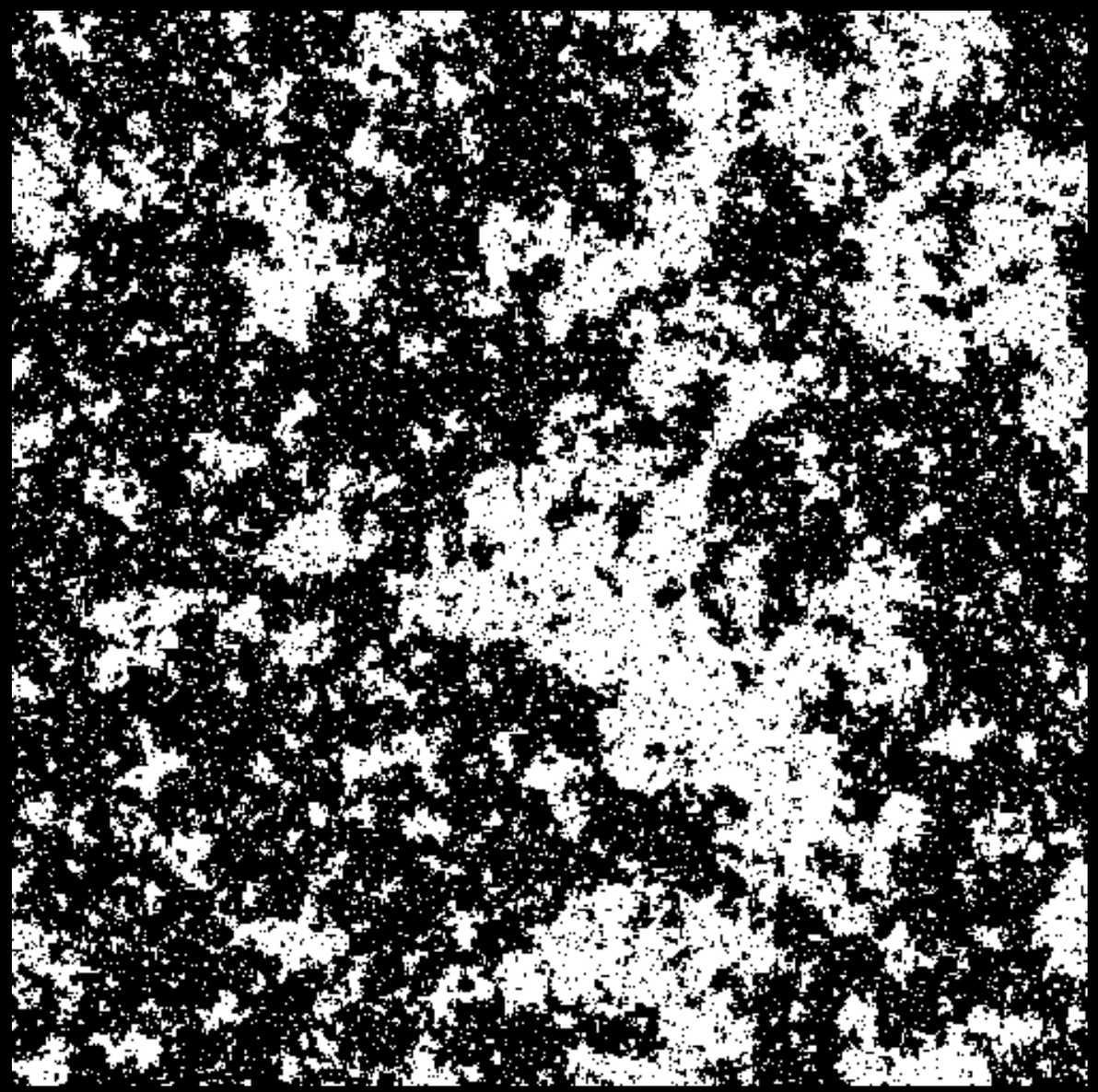
$$P(x_1, x_2, x_3, x_4) \propto f_1(x_1, x_2) f_2(x_2, x_3, x_4)$$

Ising model

- A pairwise connected factor graph with simple factors:



$$f(x, y) = \exp(c\delta_{x=y})$$



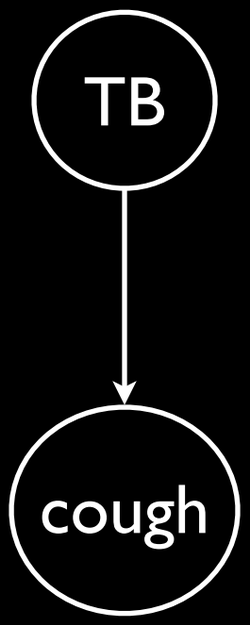
Languages for probability

- Purposes of a language:
  - Makes writing down models easier.
  - Makes reasoning about models clearer.
  - Supports efficient inference.
  - Gives ideas about mental representation.

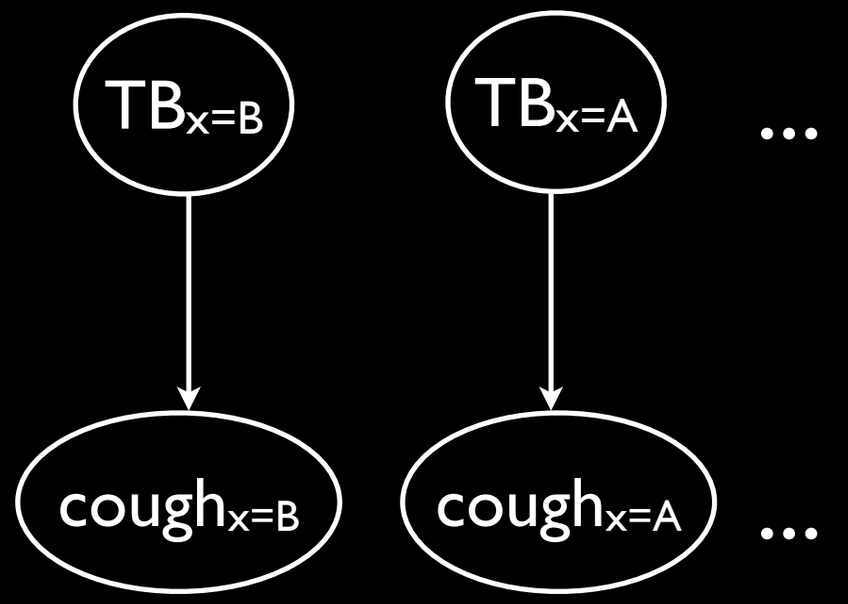
Toward finer languages

- Bayes nets are probabilistic propositional logic... isn't that enough?
- No:
  - Fixed set of variables.
  - Structure within variables?
  - Structure in the factors (CPTs, stochastic fns)?
  - Abstraction: there is no way in the language to make a new stochastic fn.

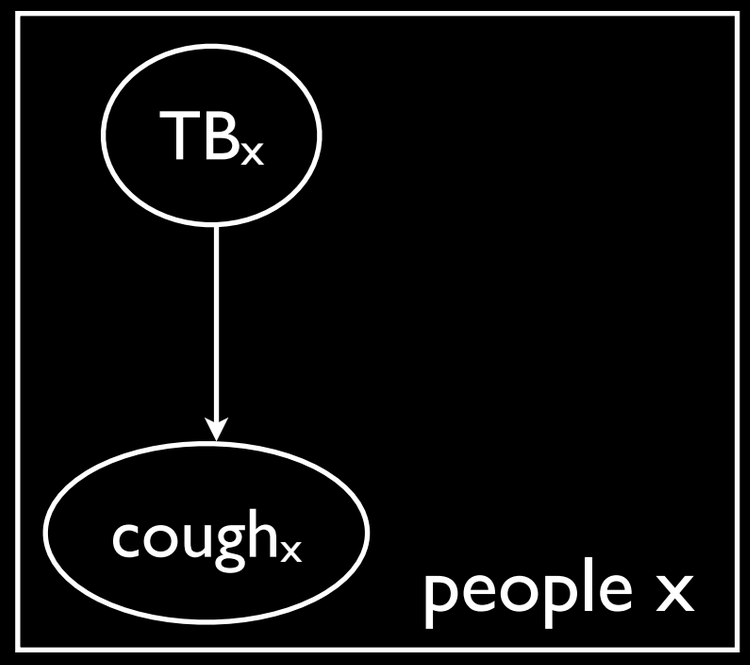
Objects



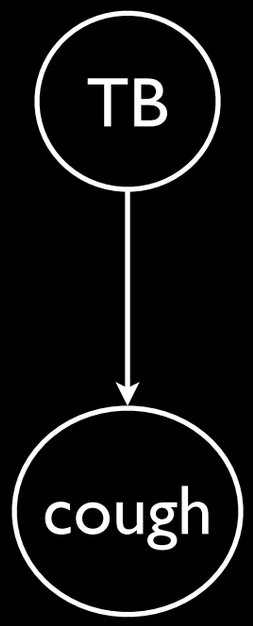
- *Bob* having TB doesn't mean *Alice* does.
- Plates: make a copy of what's inside the square for every element of the set.



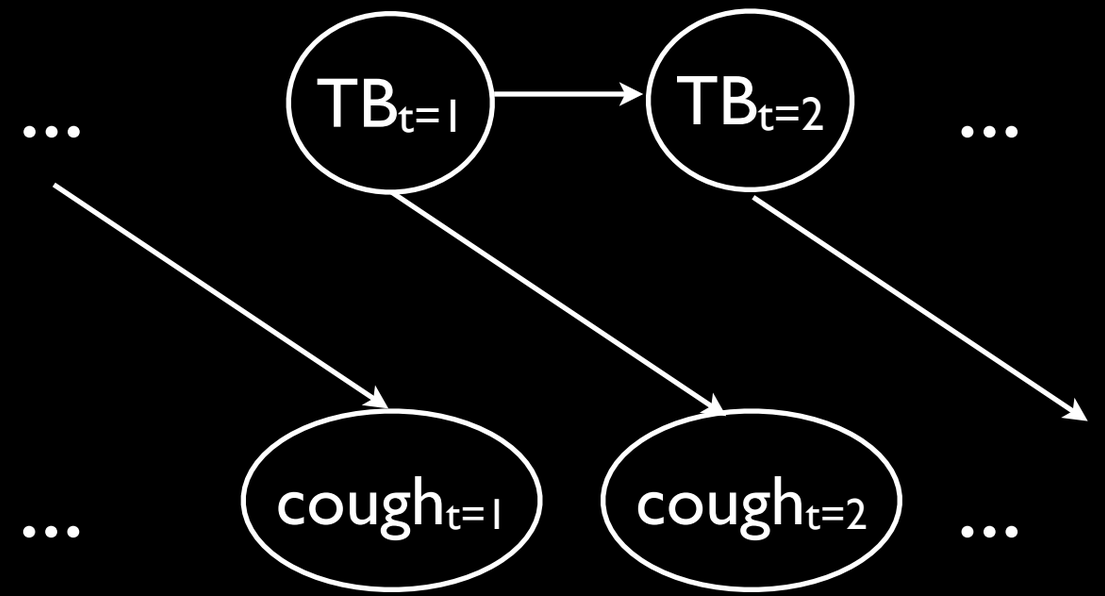
- A simple form of universal quantifier.
- What if you don't know the set ahead of time?



Dynamic BNs

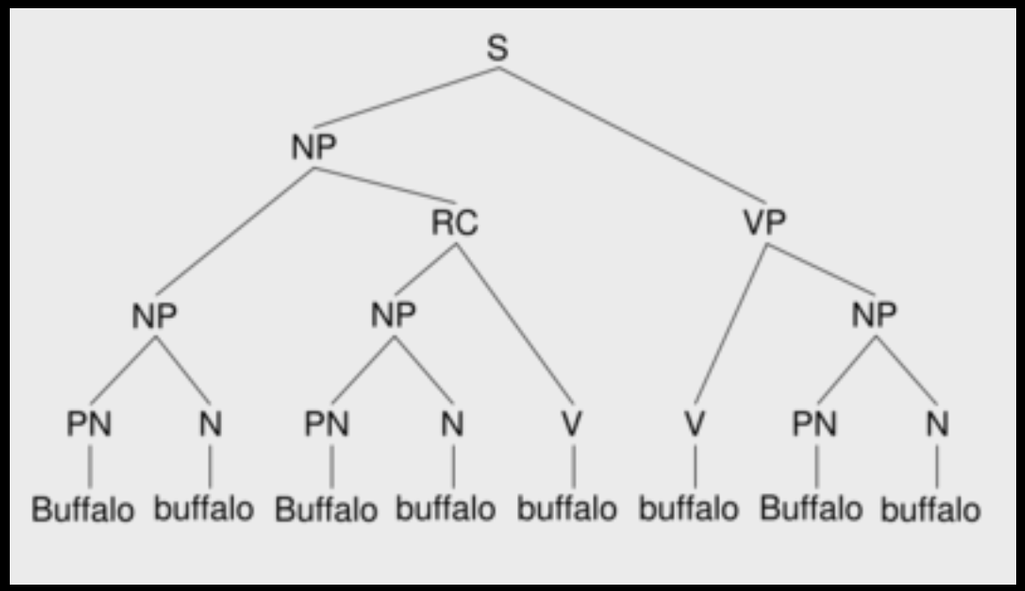


- TB *now* causes coughing *later*.



- In a DBN two time steps are understood to stand for all time.
  - Much like a plate, but not.
  - Why can't we have common notation?

PCFG



- A grammar generates structures of unbounded size and complexity. (More on this later.)
- What does the graphical model look like???

Context-specific indep.

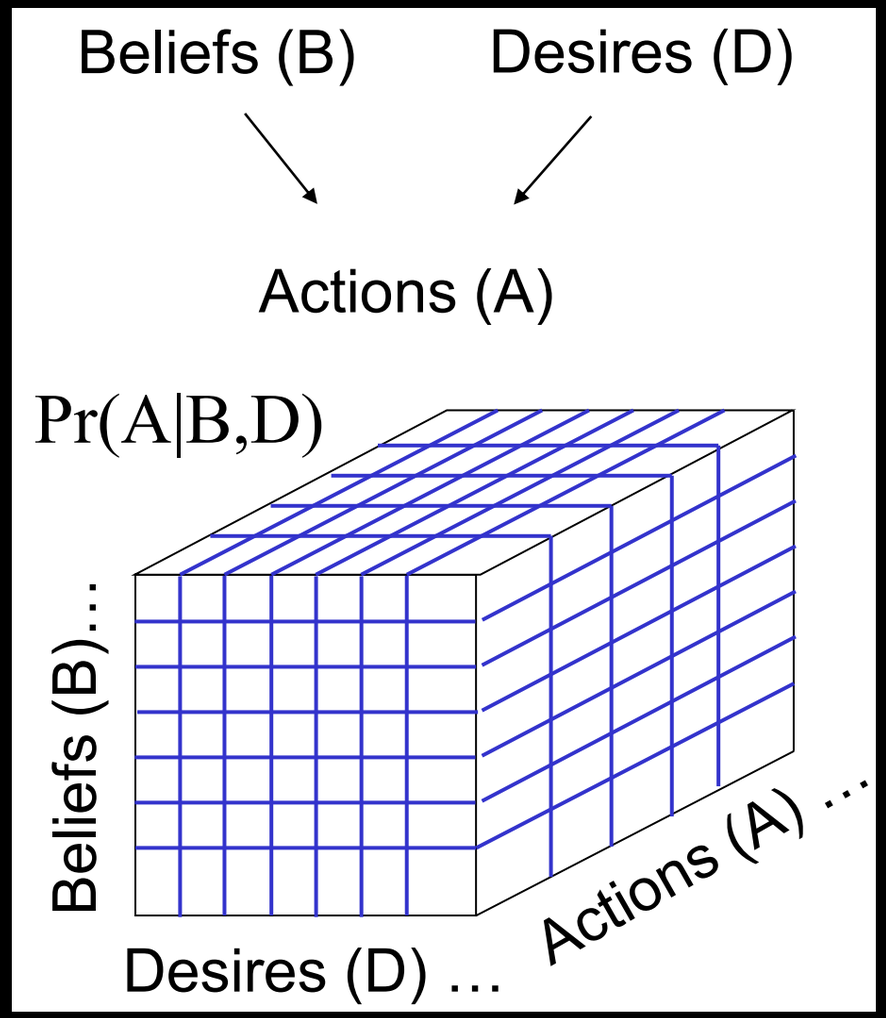
- When A=t, D is independent of B and C.
- This independence is hidden in the factor (CPT).
  - Must the functions we compose be black boxes?

A	B	C	D = t
t	t	t	0.8
t	t	f	0.8
t	f	t	0.8
t	f	f	0.8
f	t	t	0.5
f	t	f	0.7
f	f	t	0.2
f	f	f	0.1

See Boutilier, Friedman, Goldszmidt, Koller (1996).

Infinite domains

- When the domain of variables is infinite, CPTs aren't an option.
- We are building a language for describing distributions...
- Can't we use it to describe the stochastic functions we need compositionally?



Break

Languages for probability

- Purposes of a language:
  - Makes writing down models easier.
  - Makes reasoning about models clearer.
  - Supports efficient inference.
  - Gives ideas about mental representation.

"graphical models" is more than just graphs... pictures, math, and words... into the same language?

Higher-order probability

- How can we create a language for probabilities more fine-grained than Bayes nets?
  - Undirected methods: weights on predicate logic clauses. (See Domingos saturday!)
  - Directed methods: exploit the control flow of a higher-order logic -- a programming language. (Programs are causal instructions.)
    - Imperative: familiar, but logical basis, and causality obscure.
    - Functional: initially hard, but brings many powerful representation ideas.

λ calculus

λ calculus

- Notation:
  - Function have parentheses on the wrong side: `(sin x)`
  - Operators always go at the beginning: `(+ x y)`

λ calculus

- Notation:
  - Function have parentheses on the wrong side: `(sin x)`
  - Operators always go at the beginning: `(+ x y)`
- λ makes functions, define binds values to symbols:

```
(define double
  (λ (x) (+ x x)))
```

λ calculus

- Notation:
  - Function have parentheses on the wrong side: `(sin x)`
  - Operators always go at the beginning: `(+ x y)`
- λ makes functions, define binds values to symbols:

```
(define double
  (λ (x) (+ x x)))
```

```
(double 3)   => 6
```

λ calculus

- Notation:
  - Function have parentheses on the wrong side: `(sin x)`
  - Operators always go at the beginning: `(+ x y)`
- λ makes functions, define binds values to symbols:

```
(define double
  (λ (x) (+ x x)))
```

```
(double 3)   => 6
```

```
(define repeat
  (λ (f) (λ (x) (f (f x)))))
```

λ calculus

- Notation:
  - Function have parentheses on the wrong side: `(sin x)`
  - Operators always go at the beginning: `(+ x y)`
- λ makes functions, define binds values to symbols:

```
(define double
  (λ (x) (+ x x)))
```

```
(double 3)   => 6
```

```
(define repeat
  (λ (f) (λ (x) (f (f x)))))
```

```
((repeat double) 3)  => 12
```

λ calculus

- Notation:
  - Function have parentheses on the wrong side: `(sin x)`
  - Operators always go at the beginning: `(+ x y)`
- λ makes functions, define binds values to symbols:

```
(define double
  (λ (x) (+ x x)))
```

```
(double 3)  => 6
```

```
(define repeat
  (λ (f) (λ (x) (f (f x)))))
```

```
((repeat double) 3)  => 12
```

```
(define 2nd-derivative (repeat derivative))
```

λ calculus

- More formally, the set of Lambda expressions L is defined by:
  - Each variable symbol $x \in L$.
  - If x is a variable and $M \in L$, then $(\lambda\ (x)\ M) \in L$.
  - If $M, N \in L$, then $(M\ N) \in L$.
- *Evaluation* is defined by the reductions:
  - α-conversion: change the name of a bound variable.
  - β-reduction: $((\lambda\ (x)\ M)\ N)$ becomes $M^{x:=N}$.
  - η-conversion: $((\lambda\ (x)\ M)\ x)$ is M.
- We assume applicative order: arguments are always reduced before functions.

ψλ-calculus

- How can we use these ideas to describe probabilities?
- ψλ-calculus: a stochastic variant.
  - We introduce a random primitive `flip`, such that `(flip)` reduces to a random sample `t/f`.
  - The usual evaluation rules now result in *sampled* values. This induces *distributions*.
- This calculus, plus primitive operators and data types, gives the probabilistic programming language Church.

Church

Random primitives:

```
(define a (flip 0.3))
(define b (flip 0.3))
(define c (flip 0.3))
(+ a b c)
```

Goodman, Mansinghka, Roy,
Bonawitz, Tenenabum (2008)

Church

Random primitives:

```
(define a (flip 0.3))
(define b (flip 0.3))
(define c (flip 0.3))
(+ a b c)
```

=> 1

Goodman, Mansinghka, Roy,
Bonawitz, Tenenabum (2008)

Church

Random primitives:

```
(define a (flip 0.3))
(define b (flip 0.3))
(define c (flip 0.3))
(+ a b c)
```

```
=> 1
=> 0
```

Goodman, Mansinghka, Roy,
Bonawitz, Tenenabum (2008)

Church

Random primitives:

```
(define a (flip 0.3)) => 1
(define b (flip 0.3)) => 0
(define c (flip 0.3)) => 1
(+ a b c)
```

Goodman, Mansinghka, Roy,
Bonawitz, Tenenabum (2008)

Church

Random primitives:

```
(define a (flip 0.3))  => 1
(define b (flip 0.3))  => 0
(define c (flip 0.3))  => 1
(+ a b c)              => 2
```

Goodman, Mansinghka, Roy,
Bonawitz, Tenenabum (2008)

Church

Random primitives:

```
(define a (flip 0.3))
(define b (flip 0.3))
(define c (flip 0.3))
(+ a b c)
```

```
=> 1 0
=> 0 0
=> 1 0
=> 2 0
```

Goodman, Mansinghka, Roy,
Bonawitz, Tenenabum (2008)

Church

Random primitives:

```
(define a (flip 0.3))
(define b (flip 0.3))
(define c (flip 0.3))
(+ a b c)
```

```
=> 1 0 0
=> 0 0 0
=> 1 0 1
=> 2 0 1
```

Goodman, Mansinghka, Roy,
Bonawitz, Tenenabum (2008)

Church

Random primitives:

```
(define a (flip 0.3))
(define b (flip 0.3))
(define c (flip 0.3))
(+ a b c)
```

```
=> 1 0 0
=> 0 0 0
=> 1 0 1
=> 2 0 1..
```

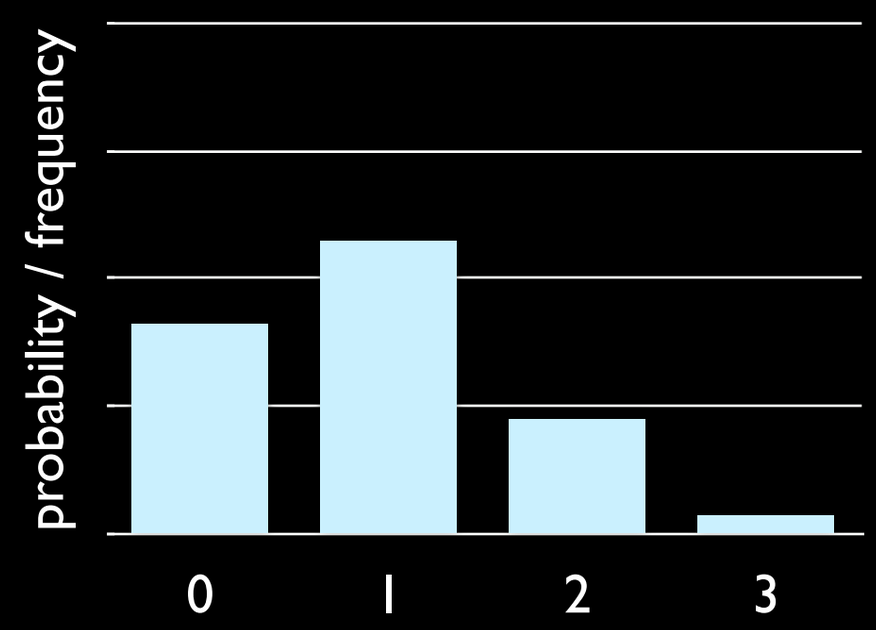
Goodman, Mansinghka, Roy,
Bonawitz, Tenenabum (2008)

Church

Random primitives:

```
(define a (flip 0.3))
(define b (flip 0.3))
(define c (flip 0.3))
(+ a b c)
```

```
=> 1 0 0
=> 0 0 0
=> 1 0 1
=> 2 0 1..
```



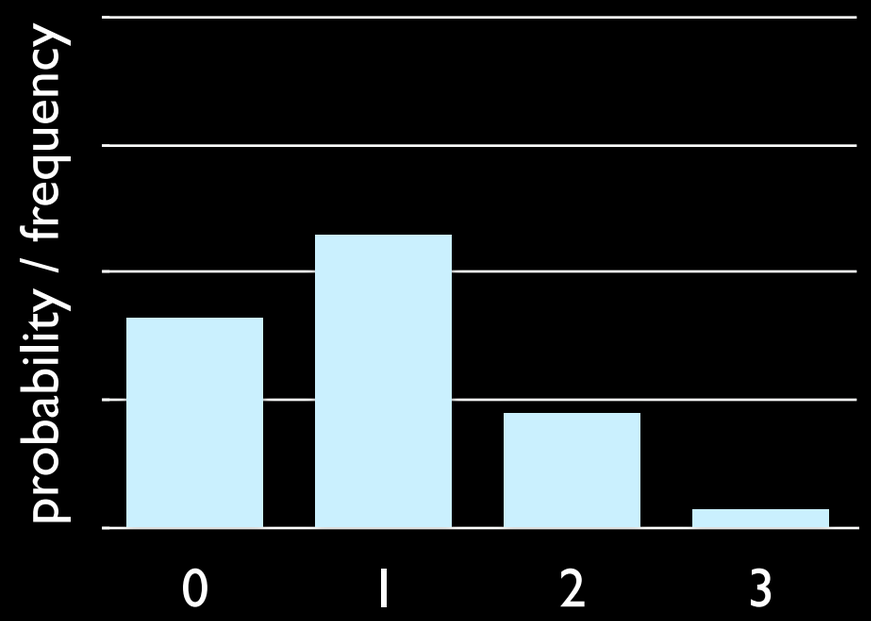
Goodman, Mansinghka, Roy, Bonawitz, Tenenabum (2008)

Church

Random primitives:

```
(define a (flip 0.3))
(define b (flip 0.3))
(define c (flip 0.3))
(+ a b c)
```

```
=> 1 0 0
=> 0 0 0
=> 1 0 1
=> 2 0 1..
```



Theorem: Any computable distribution can be represented by a Church expression.

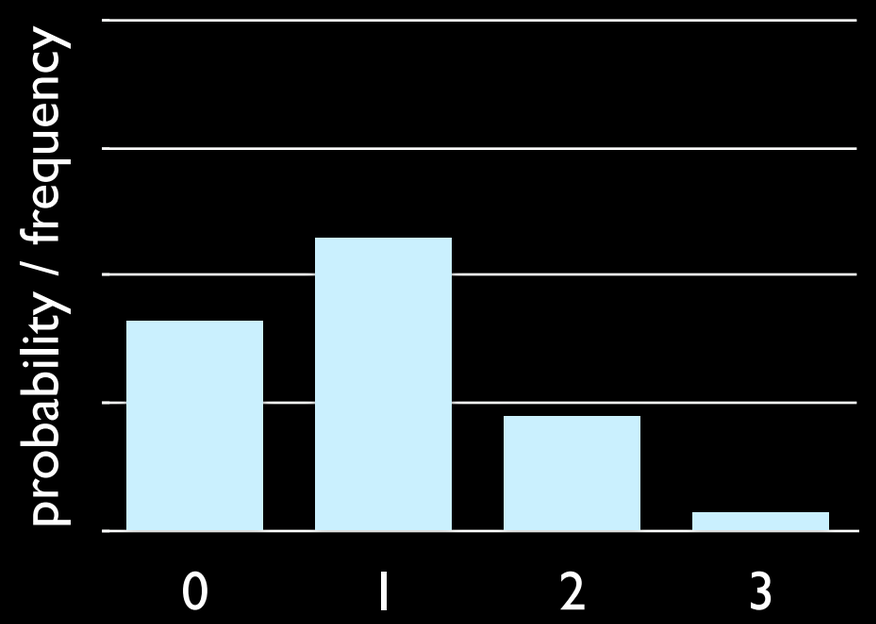
Goodman, Mansinghka, Roy, Bonawitz, Tenenabum (2008)

Church

Random primitives:

```
(define a (flip 0.3))
(define b (flip 0.3))
(define c (flip 0.3))
(+ a b c)
```

```
=> 1 0 0
=> 0 0 0
=> 1 0 1
=> 2 0 1..
```



- Compositionality is more straightforward with the sampling semantics than the distribution semantics.

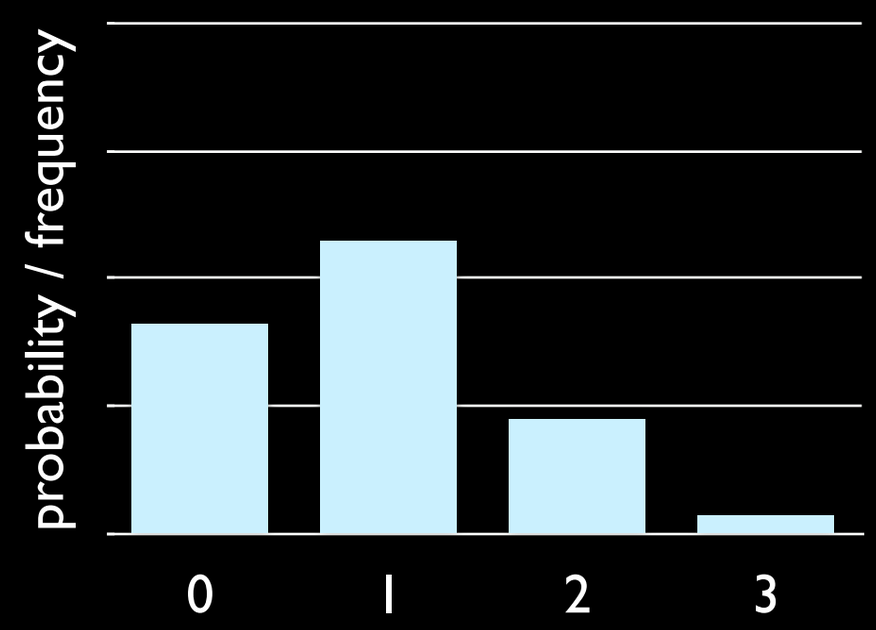
Goodman, Mansinghka, Roy, Bonawitz, Tenenabum (2008)

Church

Random primitives:

```
(define a (flip 0.3))
(define b (flip 0.3))
(define c (flip 0.3))
(+ a b c)
```

```
=> 1 0 0
=> 0 0 0
=> 1 0 1
=> 2 0 1..
```



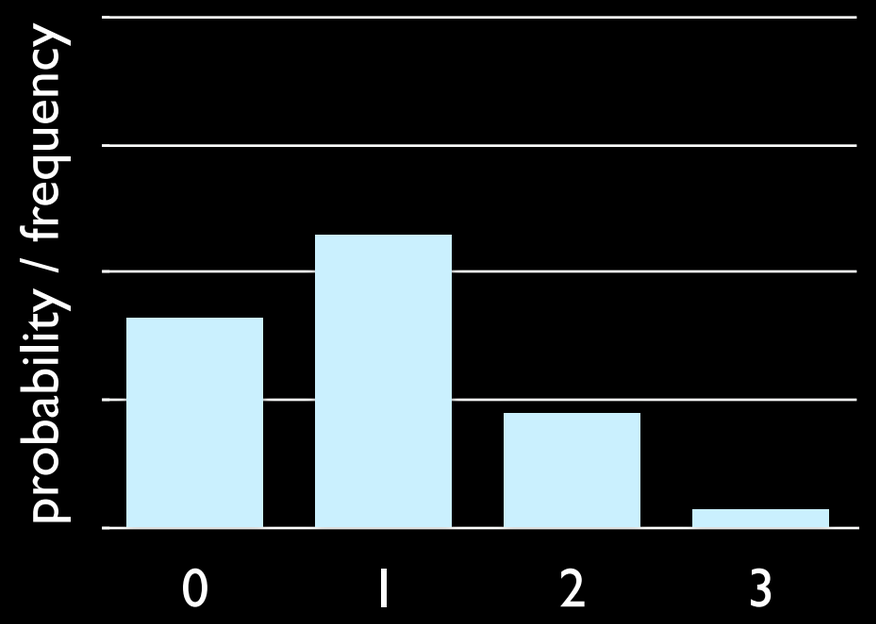
Goodman, Mansinghka, Roy, Bonawitz, Tenenabum (2008)

Church

Random primitives:

```
(define a (flip 0.3))
(define b (flip 0.3))
(define c (flip 0.3))
(+ a b c)
```

```
=> 1 0 0
=> 0 0 0
=> 1 0 1
=> 2 0 1..
```



Conditioning (inference):

```
(query
  (define a (flip 0.3))
  (define b (flip 0.3))
  (define c (flip 0.3))
  (+ a b c)
  (= (+ a b) 1))
```

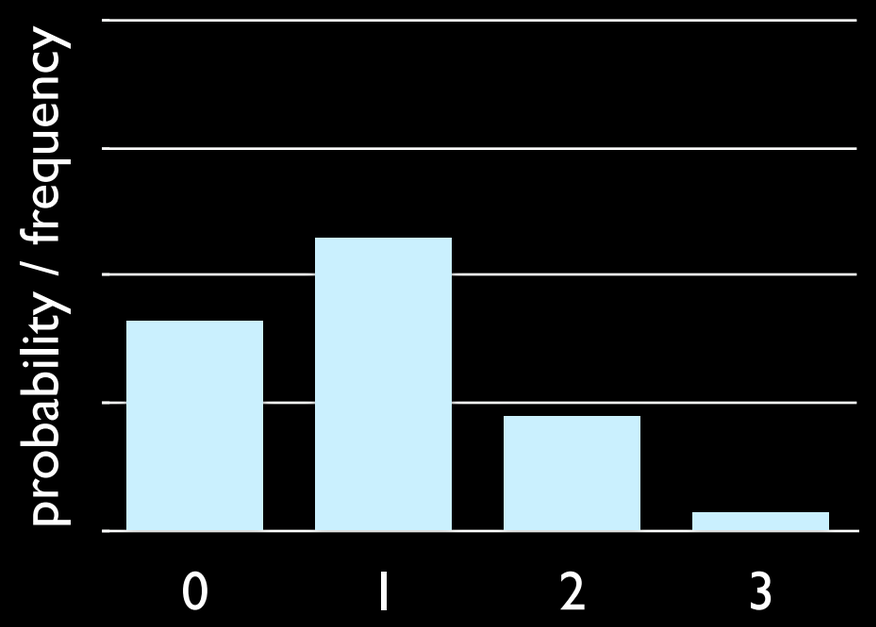
Goodman, Mansinghka, Roy, Bonawitz, Tenenabum (2008)

Church

Random primitives:

```
(define a (flip 0.3))
(define b (flip 0.3))
(define c (flip 0.3))
(+ a b c)
```

```
=> 1 0 0
=> 0 0 0
=> 1 0 1
=> 2 0 1..
```



Conditioning (inference):

```
(query
  (define a (flip 0.3))
  (define b (flip 0.3))
  (define c (flip 0.3))
  (+ a b c)                Query
  (= (+ a b) 1))
```

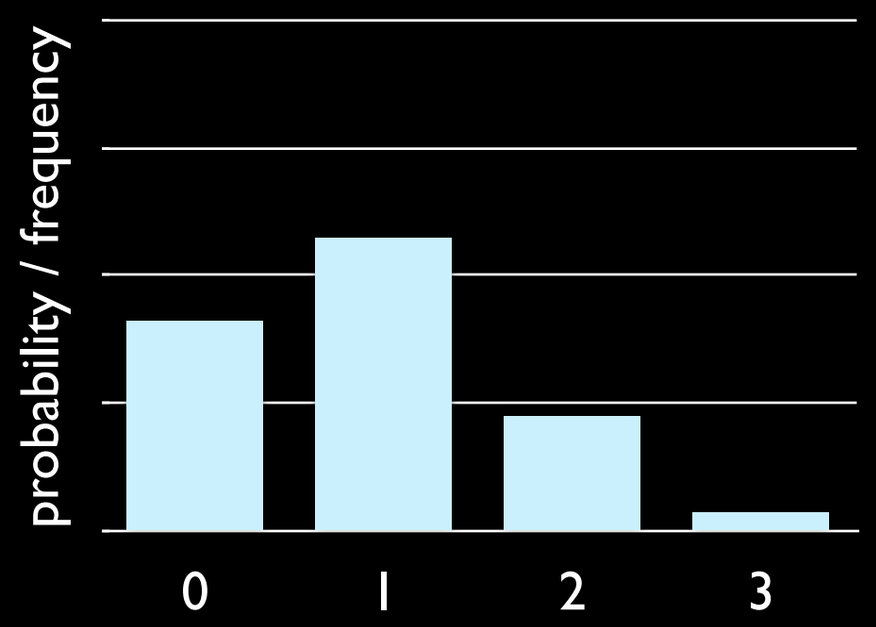
Goodman, Mansinghka, Roy, Bonawitz, Tenenabum (2008)

Church

Random primitives:

```
(define a (flip 0.3))
(define b (flip 0.3))
(define c (flip 0.3))
(+ a b c)
```

```
=> 1 0 0
=> 0 0 0
=> 1 0 1
=> 2 0 1..
```



Conditioning (inference):

```
(query
  (define a (flip 0.3))
  (define b (flip 0.3))
  (define c (flip 0.3))
  (+ a b c)
  (= (+ a b) 1))
```

Query

Condition, must be true

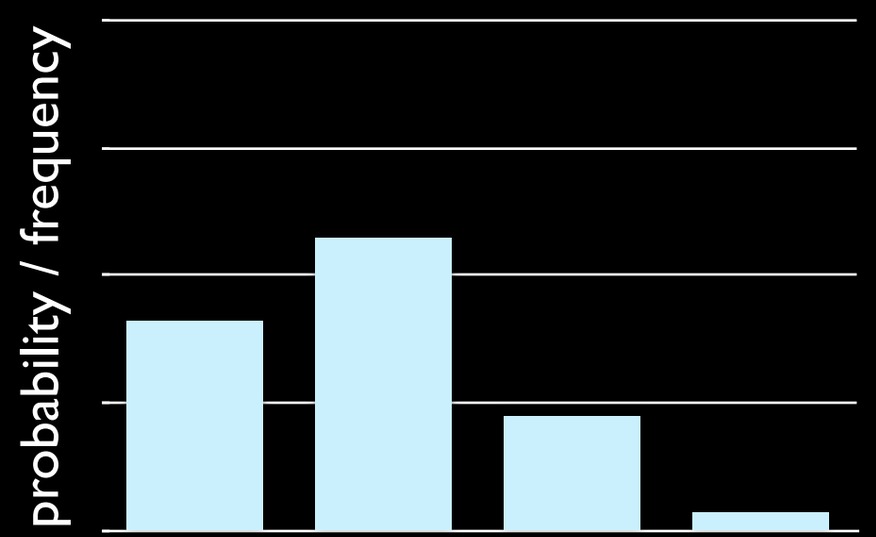
Goodman, Mansinghka, Roy, Bonawitz, Tenenabum (2008)

Church

Random primitives:

```
(define a (flip 0.3))
(define b (flip 0.3))
(define c (flip 0.3))
(+ a b c)
```

```
=> 1 0 0
=> 0 0 0
=> 1 0 1
=> 2 0 1..
```



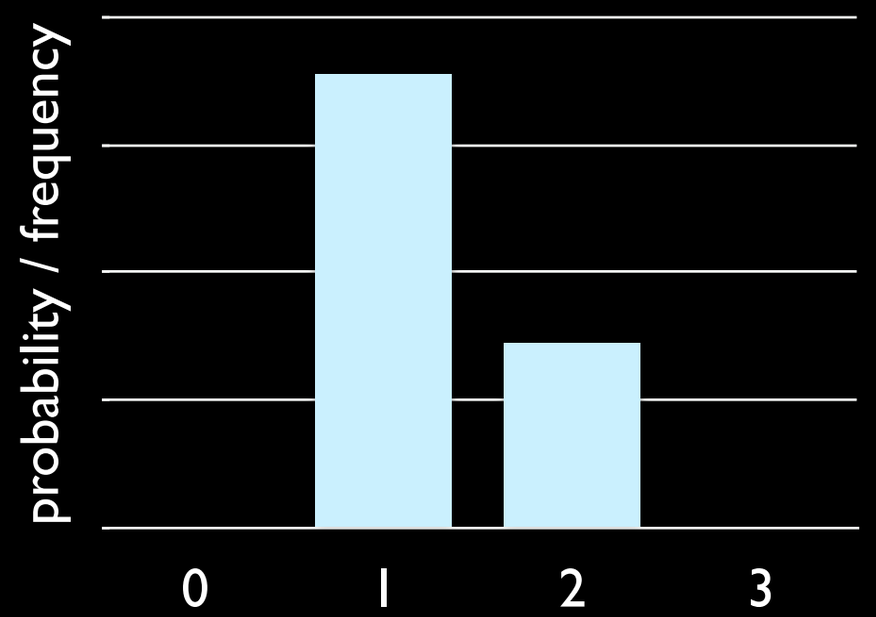
Conditioning (inference):

```
(query
  (define a (flip 0.3))
  (define b (flip 0.3))
  (define c (flip 0.3))
  (+ a b c)
  (= (+ a b) 1))
```

=>

Query

Condition, must be true



Goodman, Mansinghka, Roy, Bonawitz, Tenenabum (2008)

Church

- How can we define `query`?
  - First de-sugar into a query thunk:

```
(query
  ..defines..
  ..question..
  ..condition..)
```

→

```
(query
  (λ ()
   ..defines..
    (list ..question..
          ..condition..)))
```

  - Now define as a recursive function -- *rejection sampling*.
    - Just like "crossing out rows"...
  - Can also define directly as a conditional prob.

```
(define (query proc)
 ((λ (qv)
    (if (second qv)
        (first qv)
        (query proc))
  (proc)))
```

Inference

- Rejection is slow,
- But there are other options,
- See later lectures.

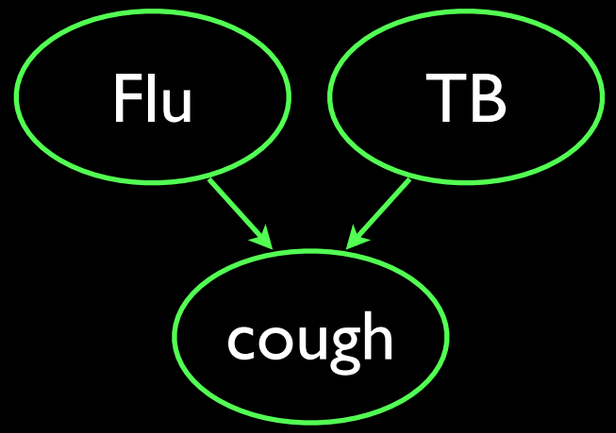
Inference *a haiku*

- Rejection is slow,
- But there are other options,
- See later lectures.

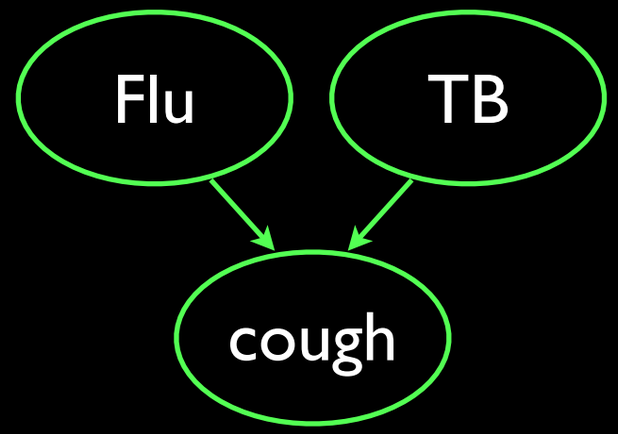
Inference

- Universal inference: an algorithm that does inference for any Church query. (And hopefully is efficient for a wide class.)
  - As a modeler, save implementation time: rapid prototyping.
  - For cognitive science, shows that the mind could be a universal inference engine.
  - Rejection is universal but slooow. But there are other options. See lectures by: Stuhlmueller, Milch, Domingos, me.

Example: BN



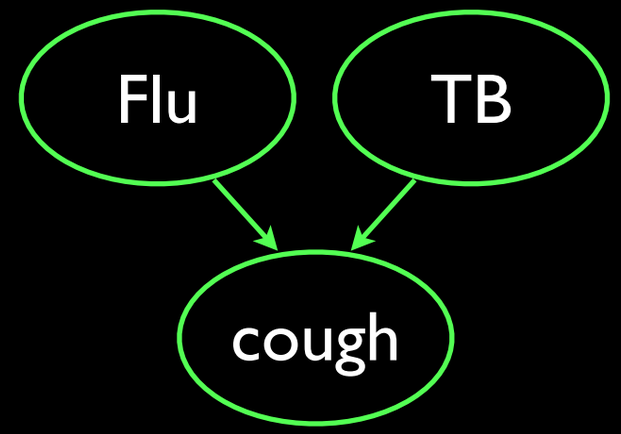
Example: BN




```
(define flu (flip 0.2))
(define TB  (flip 0.01))
(define cough
 (if (or flu TB)
     (flip 0.8) (flip 0.1)))
```

Example: BN

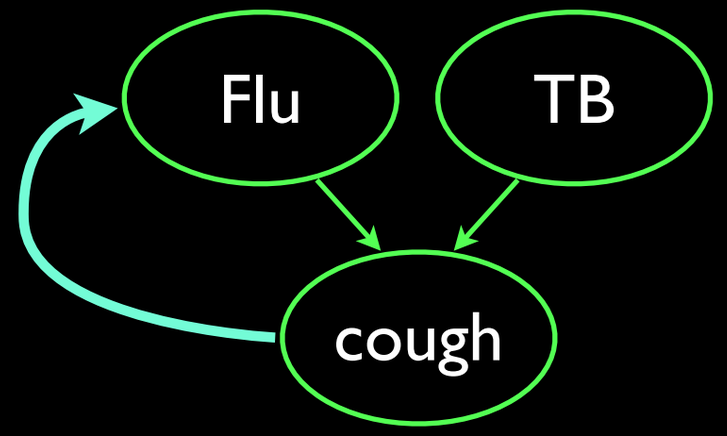
"Infer the chance of flu,
given observed cough."




```
(define flu (flip 0.2))
(define TB  (flip 0.01))
(define cough
 (if (or flu TB)
     (flip 0.8) (flip 0.1)))
```

Example: BN

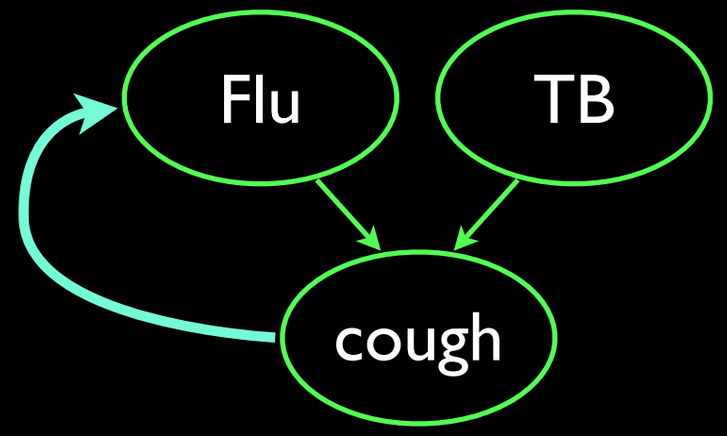
"Infer the chance of flu,
given observed cough."




```
(query
 (define flu (flip 0.2))
 (define TB  (flip 0.01))
 (define cough
  (if (or flu TB)
      (flip 0.8) (flip 0.1)))
 flu
 cough)
```

Example: BN

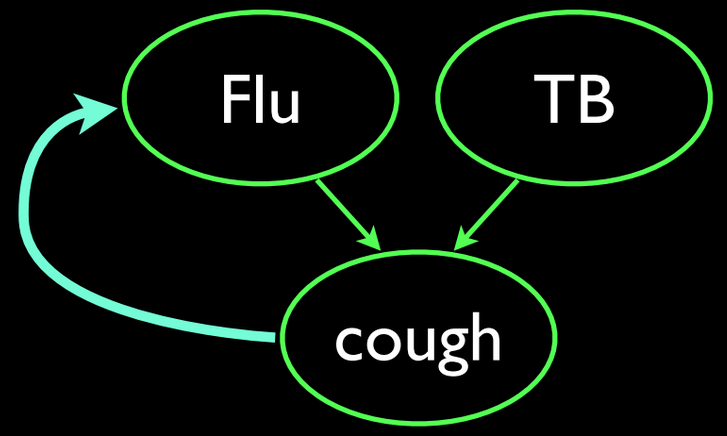
"Infer the chance of flu,
given observed cough."




```
(query
 (define flu (flip 0.2))
 (define TB  (flip 0.01))
 (define cough                          => true 66%
  (if (or flu TB)
      (flip 0.8) (flip 0.1)))
 flu
 cough)
```

Example: BN

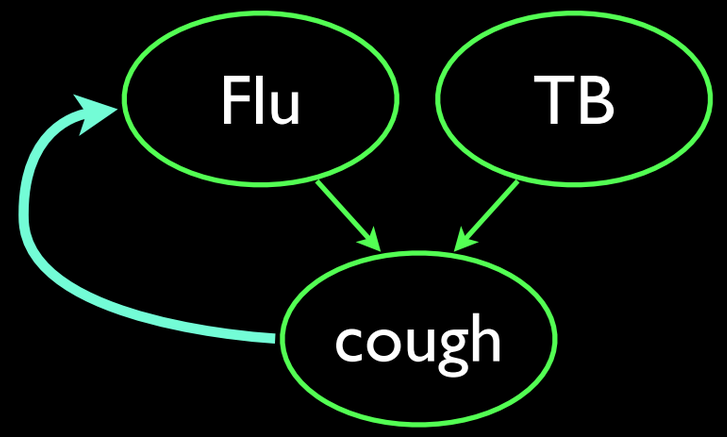
"Infer the chance of flu,
given observed cough."




```
(query
 (define flu (flip 0.2))
 (define TB  (flip 0.01))
 (define cough                          => true 66%
  (if (or flu TB)
      (flip 0.8) (flip 0.1)))
 flu
 (and cough TB))
```

Example: BN

"Infer the chance of flu,
given observed cough."




```
(query
 (define flu (flip 0.2))
 (define TB  (flip 0.01))
 (define cough                          => true 20%
  (if (or flu TB)
      (flip 0.8) (flip 0.1)))
 flu
 (and cough TB))
```

Example: objects

- To apply to different objects, variables become functions in their own right.
  - Dependence happens by *function calls*.
- The object symbols need not be specified in the model.
  - Cf. plates...

```
(define flu (λ (x) (flip 0.2)))
(define TB (λ (x) (flip 0.01)))
(define cough
 (λ (x)
  (if (or (flu x) (TB x))
      (flip 0.8) (flip 0.1))))
(cough ‘bob)
```

Example: objects

- But if we ask whether bob is coughing twice (in the same "world") we get different answers.

```
(and (cough 'bob) (cough 'bob))  => true 26%
```

- Solution: `mem`, transforms a procedure into one that is sampled only the first time it is evaluated.

```
(define flu (mem (λ (x) (flip 0.2))))
(define TB (mem (λ (x) (flip 0.01))))
(define cough
 (mem (λ (x)
        (if (or (flu x) (TB x))
            (flip 0.8) (flip 0.1)))))
(cough 'bob)
```

Aside

- Church is defined with a *random evaluation semantics*: unless specifically memoized functions are sampled each time.
  - This makes anonymous randomness easy and long range dependence more explicit.
- We could have chosen to make all procedures memoized by default. This is called *random world semantics*.
  - Some definitions and reasoning are easier with r.w.s. (See BLOG.)

Complex reasoning

Gerstenberg and Goodman (in prep)

Complex reasoning

strength

Gerstenberg and Goodman (in prep)

Complex reasoning

strength (magnitude)

Gerstenberg and Goodman (in prep)

Complex reasoning

strength (magnitude)

laziness

Gerstenberg and Goodman (in prep)

Complex reasoning

strength (magnitude)

laziness

pulling

Gerstenberg and Goodman (in prep)

Complex reasoning

strength (magnitude)

laziness team

pulling

Gerstenberg and Goodman (in prep)

Complex reasoning

strength (magnitude)

laziness team

pulling winner

Gerstenberg and Goodman (in prep)

Complex reasoning

strength (magnitude)

laziness team

pulling winner

```
(define strength (mem (lambda (person) (gaussian 10 3))))
```

Gerstenberg and Goodman (in prep)

Complex reasoning

strength (magnitude)

laziness team

pulling winner

```
(define strength (mem (lambda (person) (gaussian 10 3))))
(define lazy (lambda (person) (flip 0.1)))
```

Gerstenberg and Goodman (in prep)

Complex reasoning

strength (magnitude)

laziness team

pulling winner

```
(define strength (mem (lambda (person) (gaussian 10 3))))
(define lazy (lambda (person) (flip 0.1)))
(define (pulling person) (if (lazy person)
                             (/ (strength person) 2)
                             (strength person)))
```

Gerstenberg and Goodman (in prep)

Complex reasoning

strength (magnitude)

laziness team

pulling winner

```
(define strength (mem (lambda (person) (gaussian 10 3))))
(define lazy (lambda (person) (flip 0.1)))
(define (pulling person) (if (lazy person)
                             (/ (strength person) 2)
                             (strength person)))
(define (total-pulling team) (sum (map pulling team)))
```

Gerstenberg and Goodman (in prep)

Complex reasoning

strength (magnitude)

laziness team

pulling winner

```
(define strength (mem (lambda (person) (gaussian 10 3))))
(define lazy (lambda (person) (flip 0.1)))
(define (pulling person) (if (lazy person)
                             (/ (strength person) 2)
                             (strength person)))
(define (total-pulling team) (sum (map pulling team)))
(define (winner team1 team2)
 (if (> (total-pulling team1) (total-pulling team2))
     team1
     team2))
```

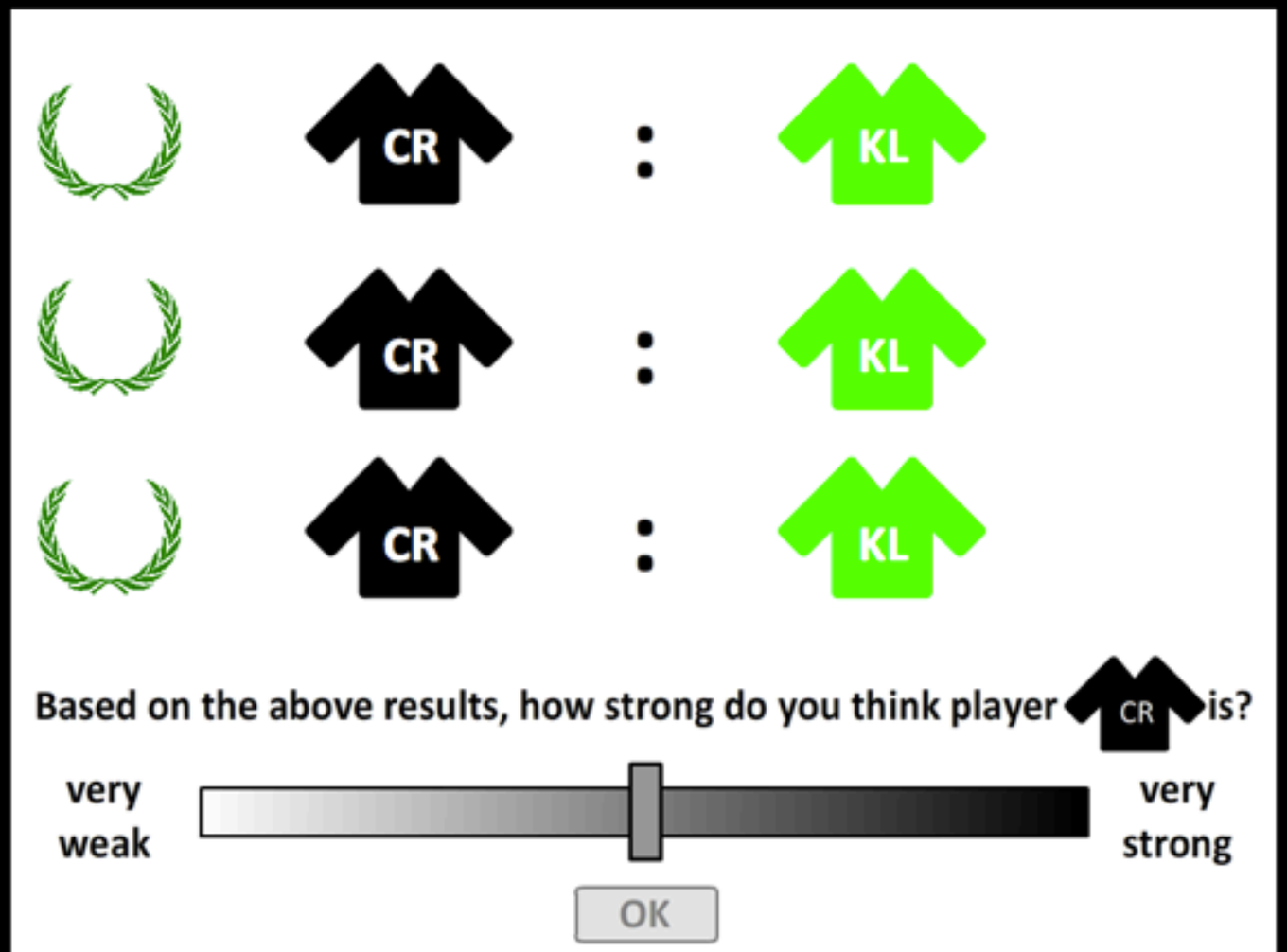
Gerstenberg and Goodman (in prep)

Complex reasoning

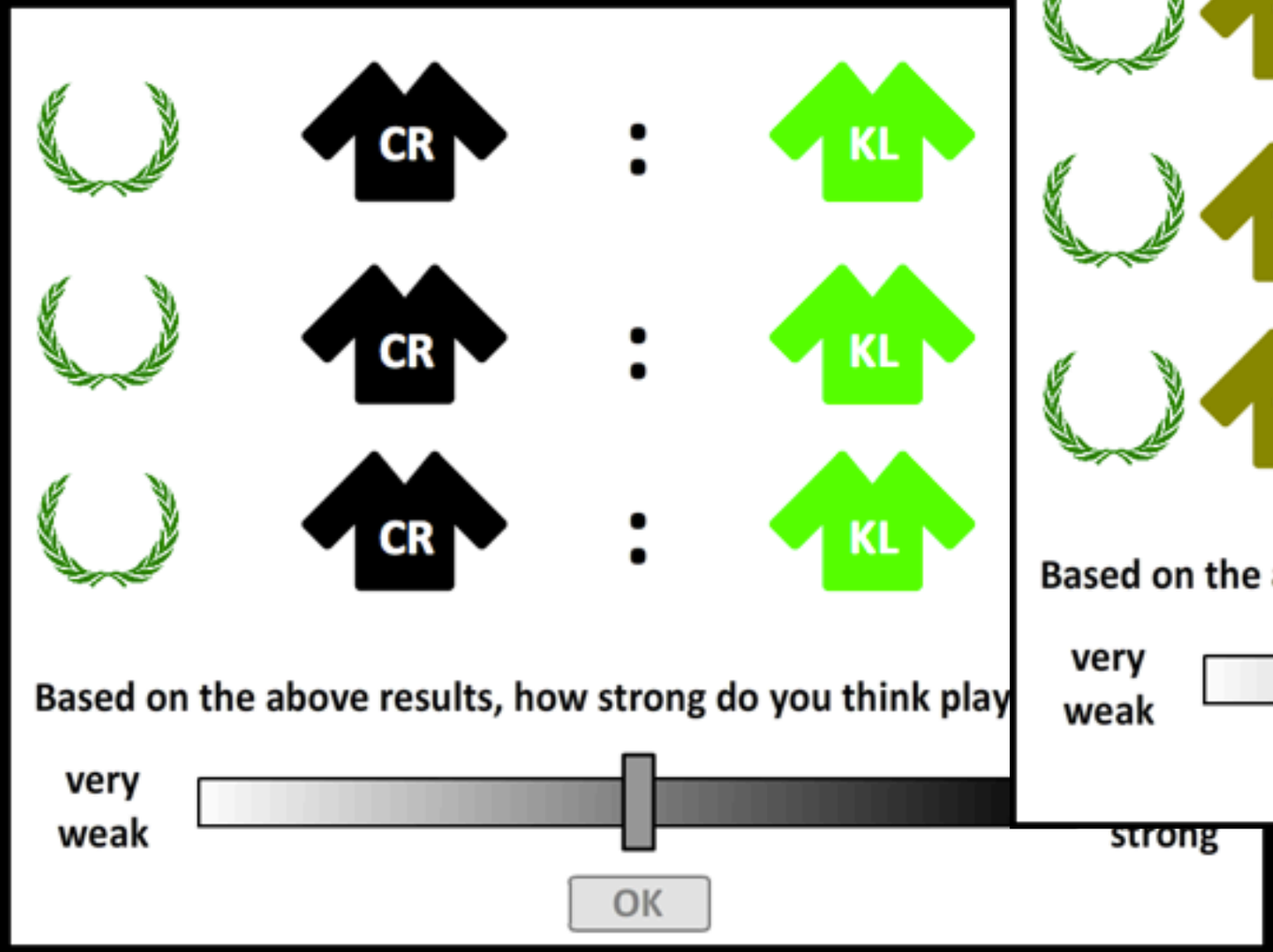
```
(query
 (define strength (mem (lambda (person) (gaussian 10 3))))
 (define (lazy person) (flip 0.1))
 (define (pulling person) (if (lazy person)
                              (/ (strength person) 2)
                              (strength person)))
 (define (total-pulling team) (sum (map pulling team)))
 (define (winner team1 team2)
  (if (> (total-pulling team1) (total-pulling team2))
      team1
      team2))
 (strength 'bob)
 (and (equal? '(bob mary) (winner '(bob mary) '(tom sue)))
      (equal? '(bob bev) (winner '(bob bev) '(jane jim)))))
```

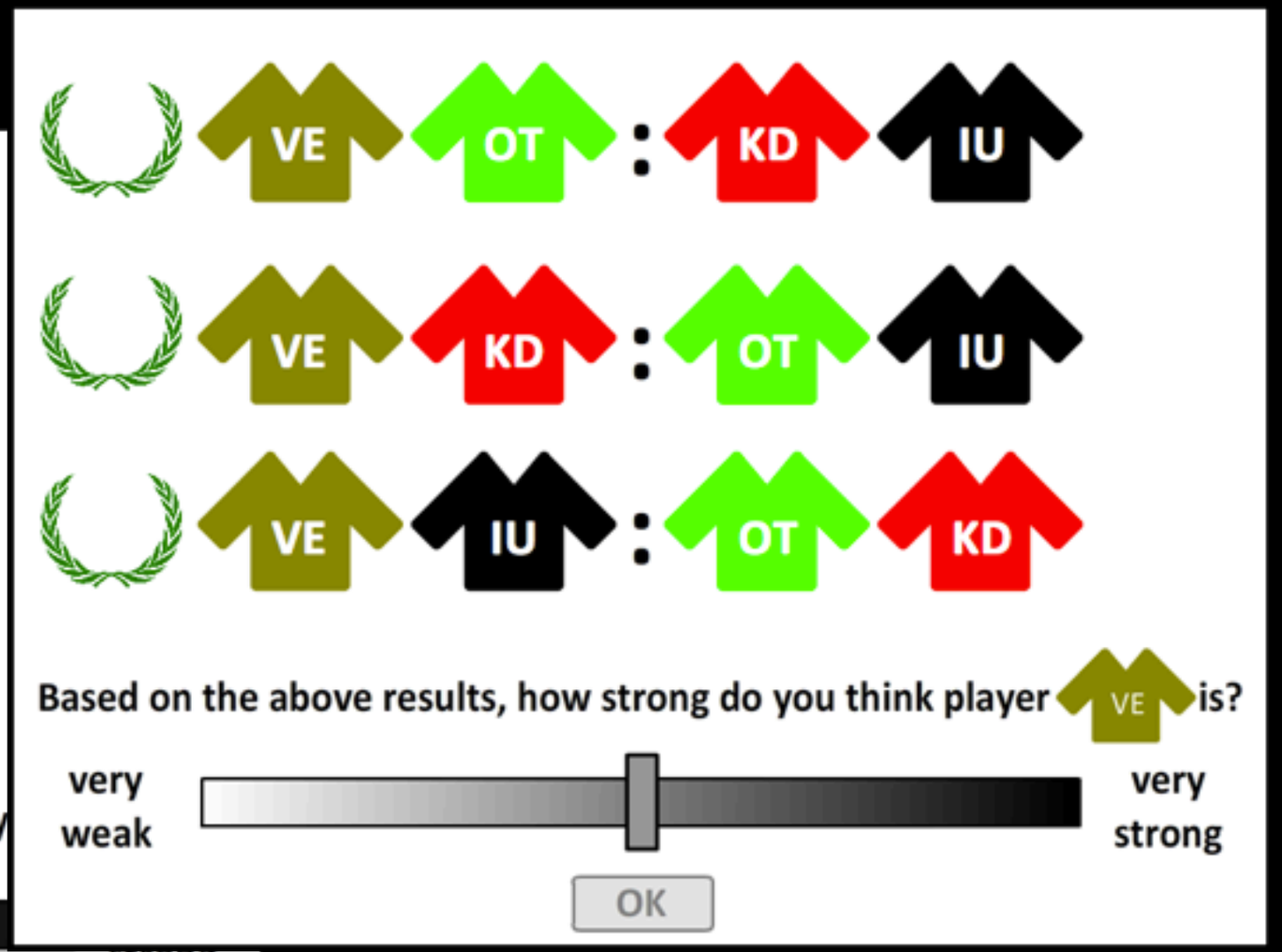
Complex reasoning

Complex reasoning



Complex reasoning



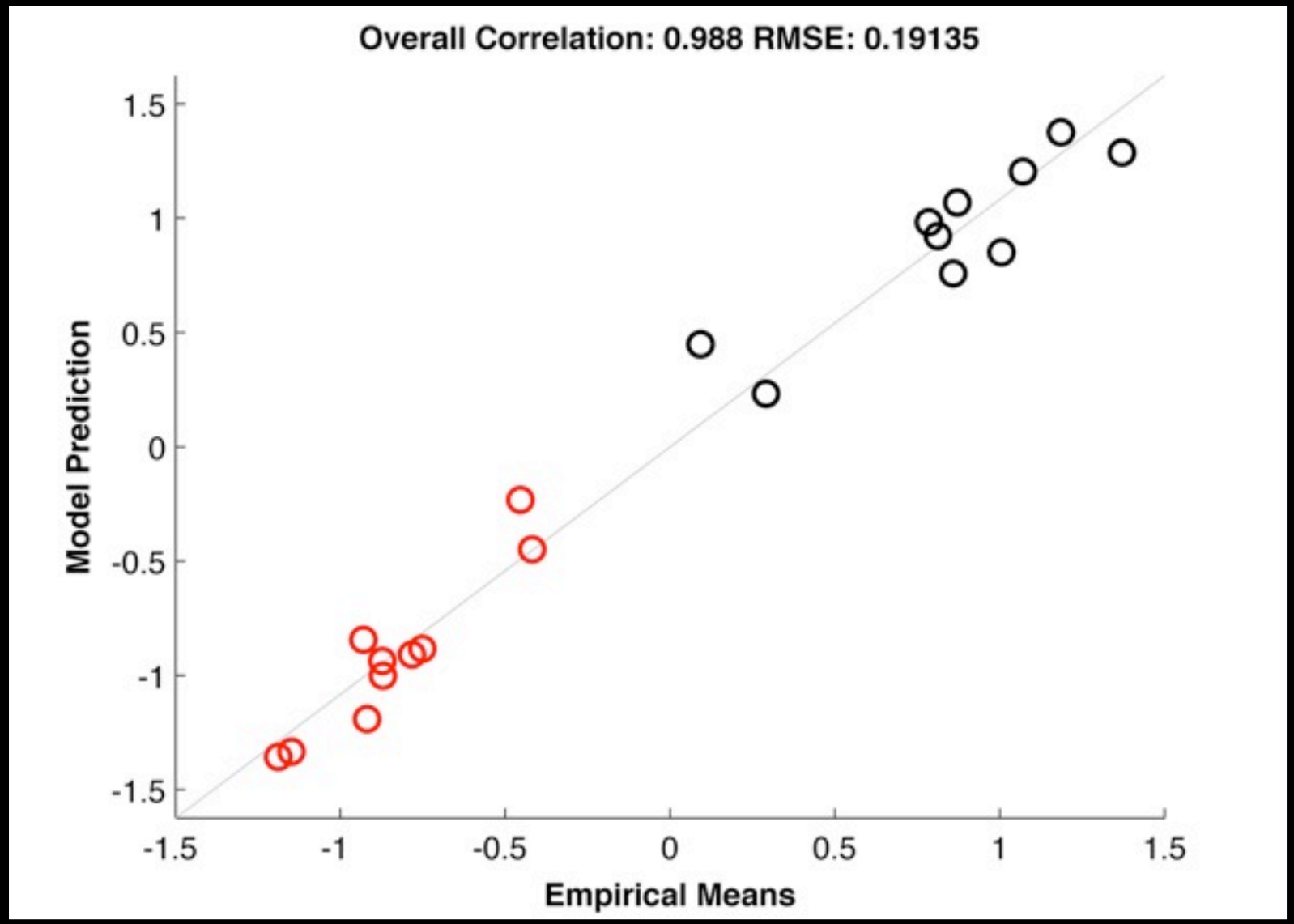


Complex reasoning

- 20 conditions: 8 single player tournaments, 12 doubles. (Within subjects design.)
- 30 participants. (Via MTurk.)

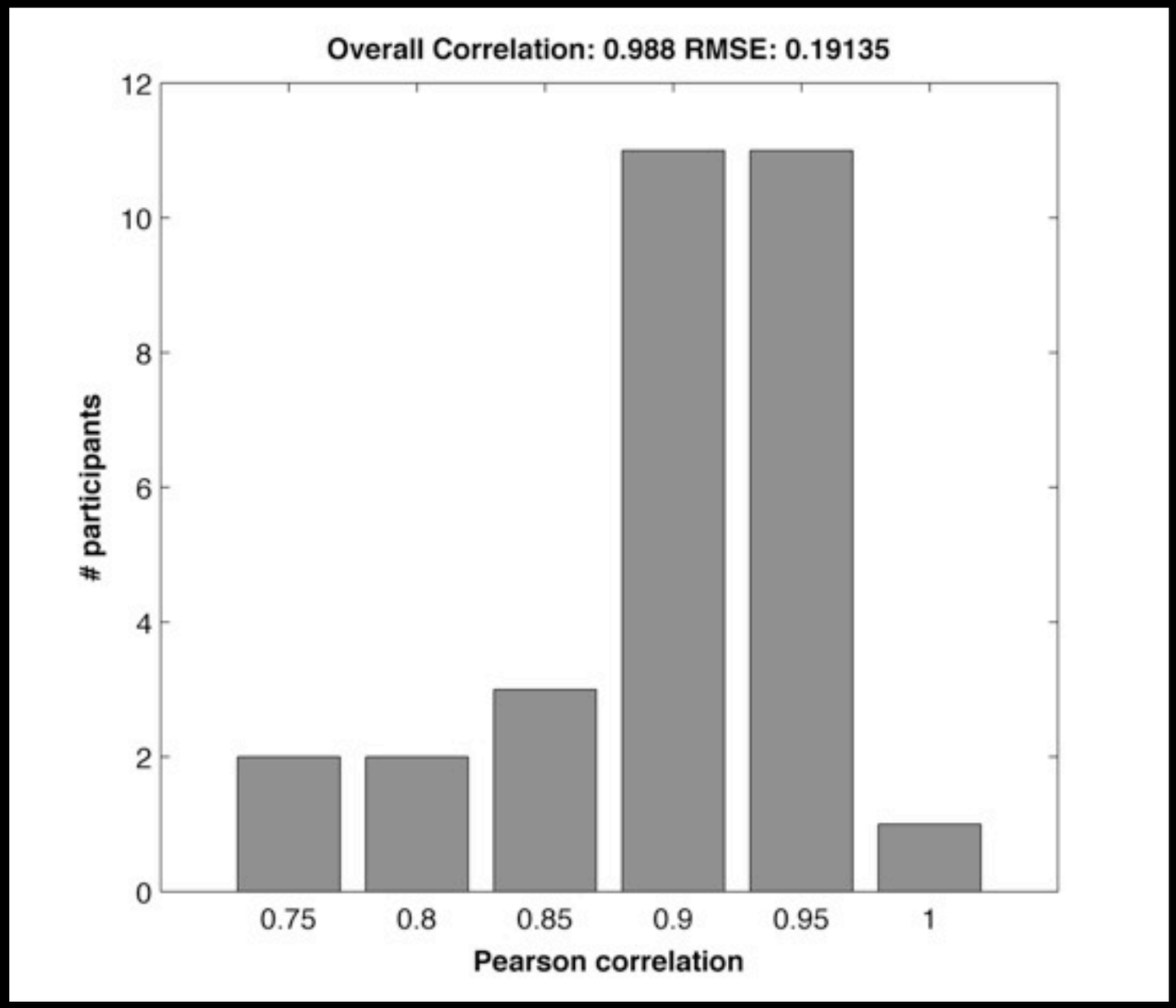
Complex reasoning

- 20 conditions: 8 single player tournaments, 12 doubles. (Within subjects design.)
- 30 participants. (Via MTurk.)



Complex reasoning

Correlation of individual participant's judgements to model predictions:



Complex reasoning

```
(query
 (define strength (mem (lambda (person) (gaussian 1 0.2))))
 (define (lazy person) (flip 0.3))
 (define (pulling person) (if (lazy person)
                              (/ (strength person) 2)
                               (strength person)))
 (define (total-pulling team) (sum (map pulling team)))
 (define (winner team1 team2)
  (if (> (total-pulling team1) (total-pulling team2))
      team1
      team2))
 (strength bob)
 (equal? '(bob mary) (winner '(bob mary) '(tom sue)))))
```

Complex reasoning

```
(query
 (define strength (mem (lambda (person) (gaussian 1 0.2))))
 (define (lazy person) (flip 0.3))
 (define (pulling person) (if (lazy person)
                              (/ (strength person) 2)
                               (strength person)))
 (define (total-pulling team) (sum (map pulling team)))
 (define (winner team1 team2)
  (if (> (total-pulling team1) (total-pulling team2))
      team1
      team2))
 (strength bob)
 (and (equal? '(bob mary) (winner '(bob mary) '(tom sue)))
      (> (strength tom) (strength bob))))
```

Complex reasoning

```
(query
 (define strength (mem (lambda (person) (gaussian 1 0.2))))
 (define (lazy person) (flip 0.3))
 (define (pulling person) (if (lazy person)
                              (/ (strength person) 2)
                               (strength person)))
 (define (total-pulling team) (sum (map pulling team)))
 (define (winner team1 team2)
  (if (> (total-pulling team1) (total-pulling team2))
      team1
      team2))
 (or (lazy bob) (> (strength mary) (strength sue)))
 (and (equal? '(bob mary) (winner '(bob mary) '(tom sue)))
      (> (strength tom) (strength bob)))))
```

Complex reasoning

Complex reasoning

- A small set of concepts leads to a huge set of potential inferences....

Complex reasoning

- A small set of concepts leads to a huge set of potential inferences....
- Stochastic functions **compose** productively and support graded **probabilistic** inference.

Complex reasoning

Alice **believes** jumping **causes** luck.
Alice **wants** to be lucky at the game.

Alice will jump before the game.

- A small set of concepts leads to a huge set of potential inferences....
- Stochastic functions **compose** productively and support graded **probabilistic** inference.

Complex reasoning

Alice **believes** jumping **causes** luck.
Alice **wants** to b

Alice will jump be

John **said** "some of the plants sprouted."
These plants almost always sprout.
John could not **see** all of the plants.

All of the plants sprouted.

- A small set of concepts leads to a huge set of potential inferences....
- Stochastic functions **compose** productively and support graded **probabilistic** inference.

Complex reasoning

Alice **believes** jumping **causes** luck.
Alice **wants** to b

Alice will jump be

John **said** "some of the plants sprouted."
These plants almost always sprout.
John could not **see** all of the plants.

All of the plants sprouted

See my lecture next Thursday.

- A small set of concepts leads to a huge set of potential inferences....
- Stochastic functions **compose** productively and support graded **probabilistic** inference.

Abstraction

```
(query
 (define face (sample-triangulation))
 (define image (render face))
 face
 (equal? observed (noisy image)))
```

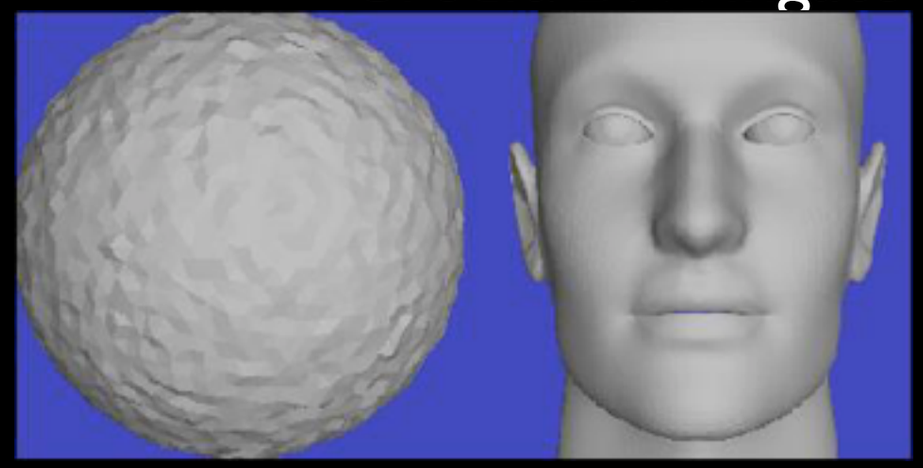
Credit: D. Wingate

- Vision as *inverse graphics*.
- The `render` function is a large deterministic function, but comes "for free" because it is a program.
- Of course, inference is hard.
  (Wingate, Stuhmueller, Siskind, and Goodman; under review)

Abstraction

```
(query
 (define face (sample-triangulation))
 (define image (render face))
 face
 (equal? observed (noisy image)))
```

Credit: D. Wingate



- Vision as *inverse graphics*.
- The `render` function is a large deterministic function, but comes "for free" because it is a program.
- Of course, inference is hard.
  (Wingate, Stuhmueller, Siskind, and Goodman; under review)

Abstraction

- We can define new *stochastic* functions using λ, and use them within the model (they are "first class").

```
(define my-logic
  (if (flip)
    (λ (x y) (or x y (flip 0.2)))
    (λ (x y) (and x y (flip 0.1)))))
(define A (flip))
(define B (flip))
(define C (my-logic A B))
```

Abstraction

- We can define new *stochastic* functions using λ, and use them within the model (they are "first class").

```
(query
 (define my-logic
   (if (flip)
     (λ (x y) (or x y (flip 0.2)))
     (λ (x y) (and x y (flip 0.1)))))
 (define A (flip))
 (define B (flip))
 (define C (my-logic A B))
 my-logic
 (and A (not C))
```

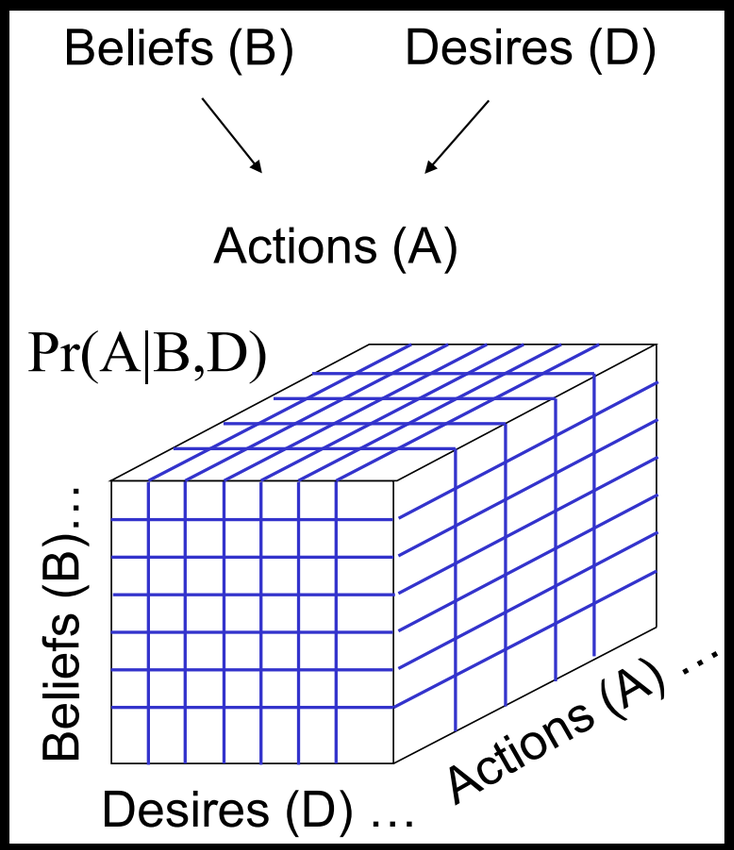
Abstraction

- We can define new *stochastic* functions using λ, and use them within the model (they are "first class").

Rule learning by inference.
See my lecture next Friday.

```
(query
 (define my-logic
   (if (flip)
     (λ (x y) (or x y (flip 0.2)))
     (λ (x y) (and x y (flip 0.1)))))
 (define A (flip))
 (define B (flip))
 (define C (my-logic A B))
 my-logic
 (and A (not C))
```

Example: decision making



Rational action as inference:

```
(define (decide state causal-model goal?)
 (query
   (define action (action-prior))
   action
   (goal?
     (causal-model state action))))))
```

Goodman, et al 2009; Baker, et al 2009.

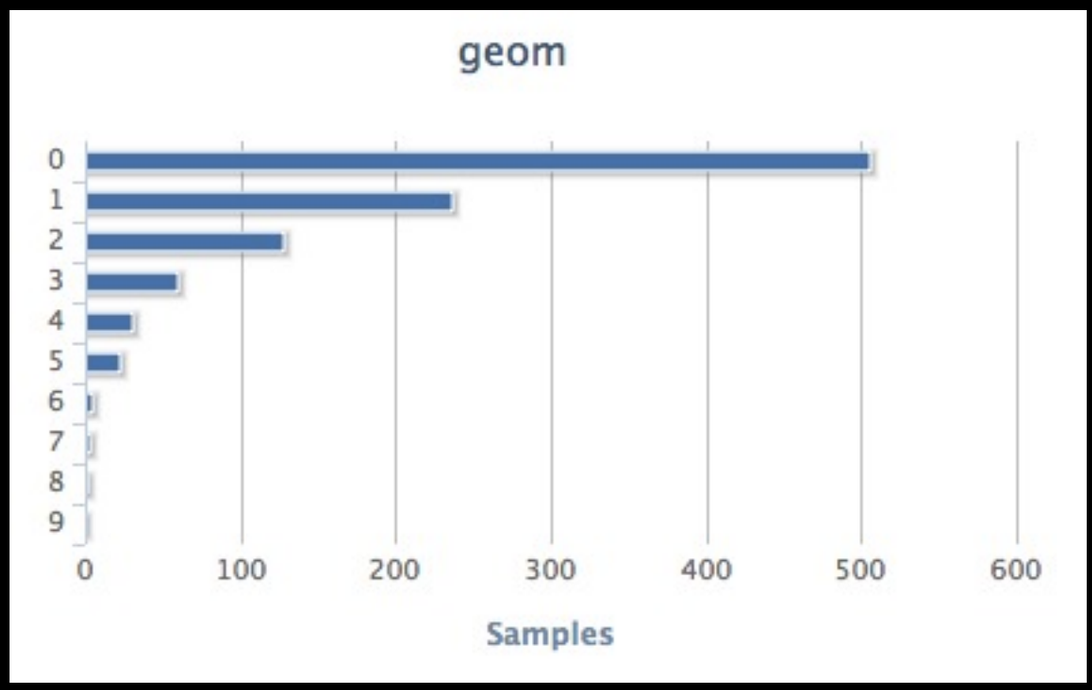
See: Lecture next thursday.

Stochastic recursion

- Stochastic recursion is a powerful way to generate worlds of unknown size.

```
(define (geom)
 (if (flip)
     0
     (+ 1 (geom))))
(repeat 1000 geom)
```

=>



Example: PCFG

- In language there is no “longest sentence”... But sentences are highly structured.
- PCFG: use stochastic recursion to structure an unbound generation.

```
(define (S)
 (if (flip 0.3)
     (list (S) (S))
     (T)))
(define (T)
 (if (flip 0.6)
     'math
     'mind))
(repeat 10 S)
```

```
=> ((math mind)
    (math mind)
    math
    mind
    (math (mind math))
    math
    ...)
```

Graphical models redux

- A *control flow analysis* of a probabilistic program is a graph representing the dependence (for evaluation) of one expression on another.
- This can never capture all independencies for all programs.
  - E.g. recursive calls may be merged into a single node (with a loop).

Conclusion 1

- Creating languages for probability distributions:
  - Makes writing down models easier (taming the Bayesian zoo).
  - Makes reasoning about models clearer.
  - Supports efficient inference.
- Gives ideas about mental representation.

	Deterministic	Stochastic
Tabular	Truth tables	Probability tables
Compose truth functions	Propositional logic	Bayes nets (factor graphs)
Compose predicates and objects	First-order logic	Markov logic
Compose arbitrary functions	λ-calculus (LISP)	ψλ-calc (Church)

Conclusion II

- The **probabilistic language of thought hypothesis**: mental representations can be thought of as expressions in ψλ-calculus.
- See my remaining lectures!
- Also see tutorial at: http://projects.csail.mit.edu/church/

wrap up with ipam context --
gonna see more "standard" graphical models for the next few days.

it's good to know they can be unified... and we'll mention it some in Q&A.