ngoodman@
stanford.edu

Church and The probabilistic language of thought

Noah D. Goodman
Stanford University

IPAM graduate summer school
July 9, 2011

Motivation

The mind is an information processing system.

Motivation

The mind is an information processing system.

But what *kind* of information processor?

Statistics and composition

Statistics and composition

Thought is useful
in an uncertain
world

Statistics and composition

Thought is useful in an uncertain world

Statistics and composition

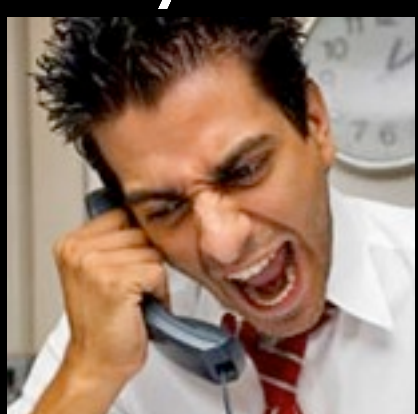
Thought is useful in an uncertain world

Why did he yell at me?

Statistics and composition

Thought is useful
in an uncertain
world

Why did he yell at me?

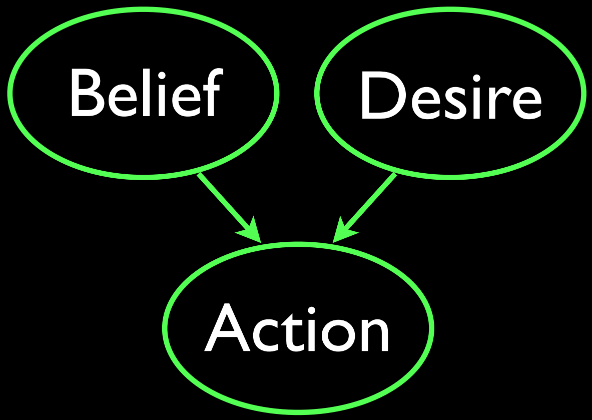


He wanted to hurt me.
He thought I was a telemarketer.

Statistics and composition

Thought is useful in an uncertain world

Why did he yell at me?

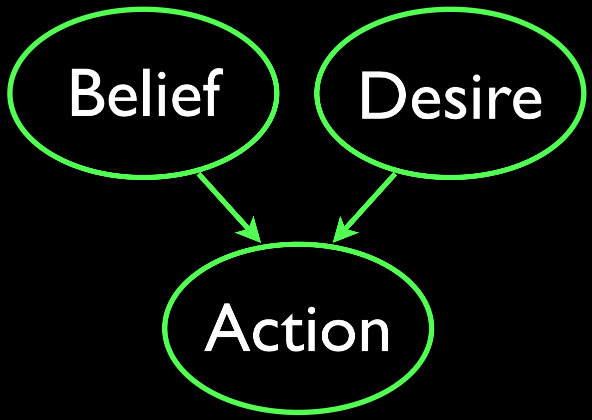


He wanted to hurt me.
He thought I was a telemarketer.

Statistics and composition

Thought is useful in an uncertain world

Why did he yell at me?

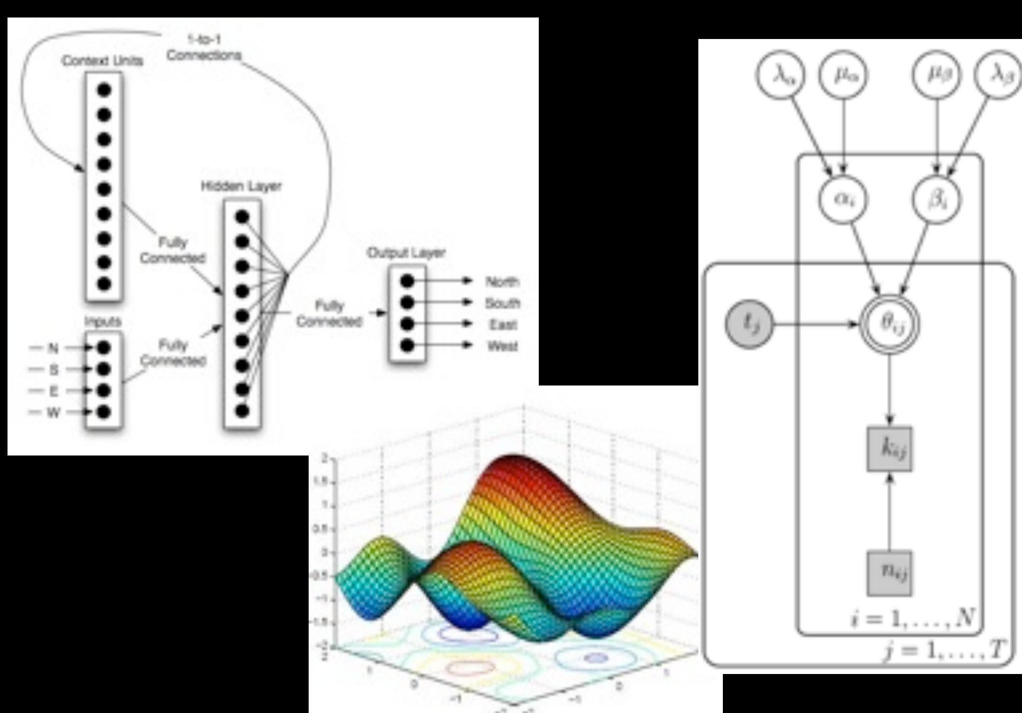


He wanted to hurt me.
He thought I was a telemarketer.

Probabilistic inference

Statistics and composition

Thought is useful in an uncertain world

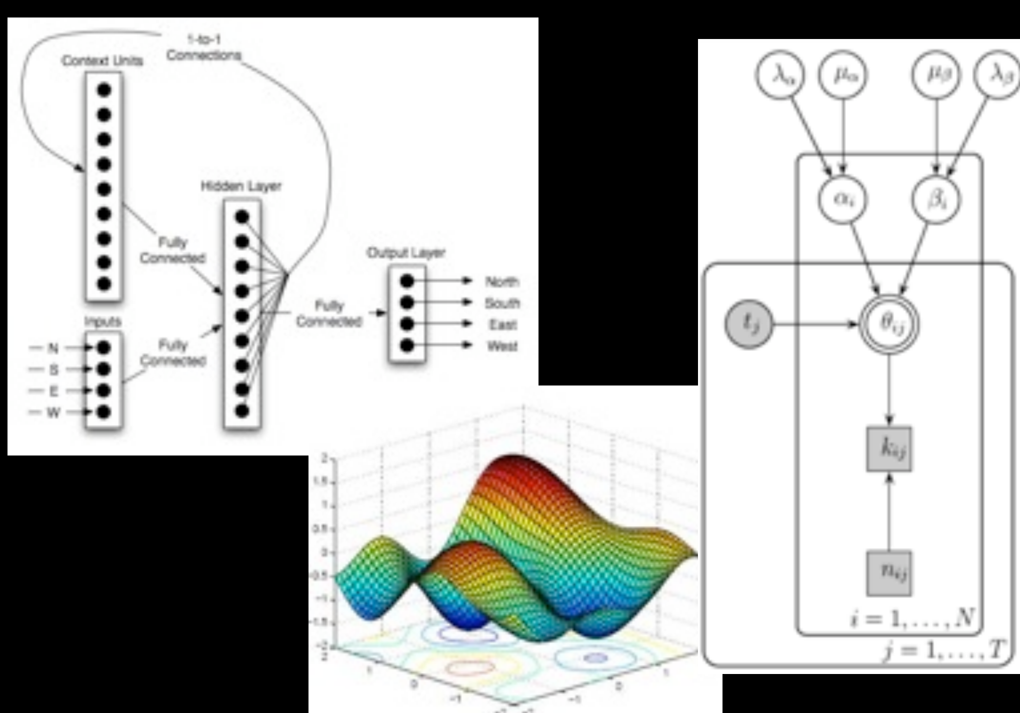


Probabilistic inference

Statistics and composition

Thought is useful in an uncertain world

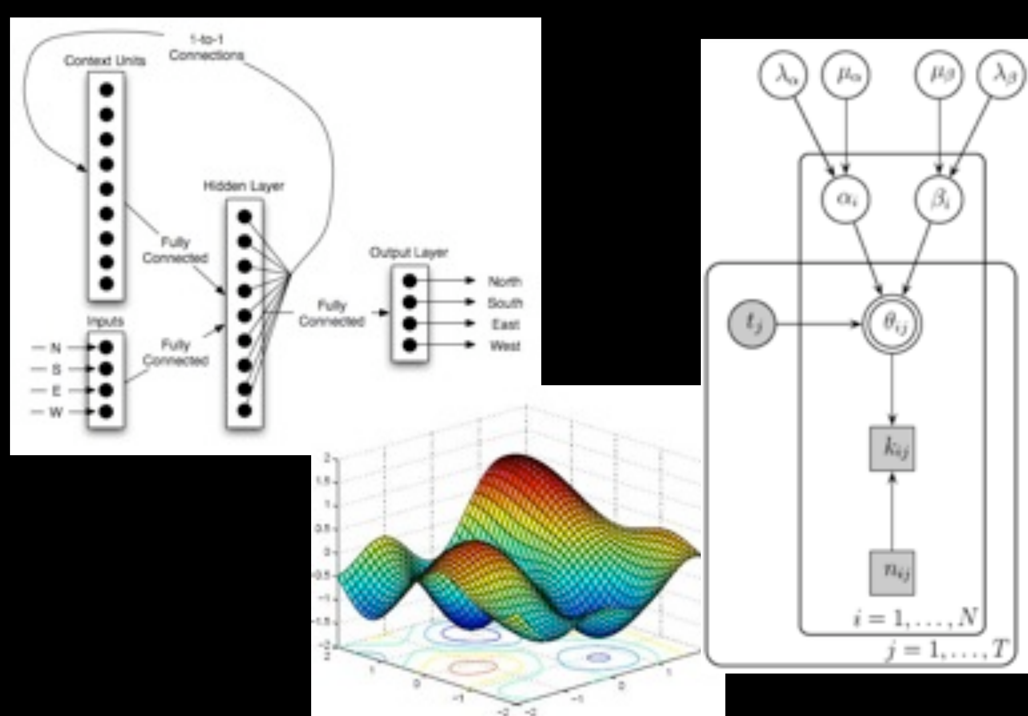
Thought is productive: “the infinite use of finite means”



Probabilistic inference

Statistics and composition

Thought is useful in an uncertain world

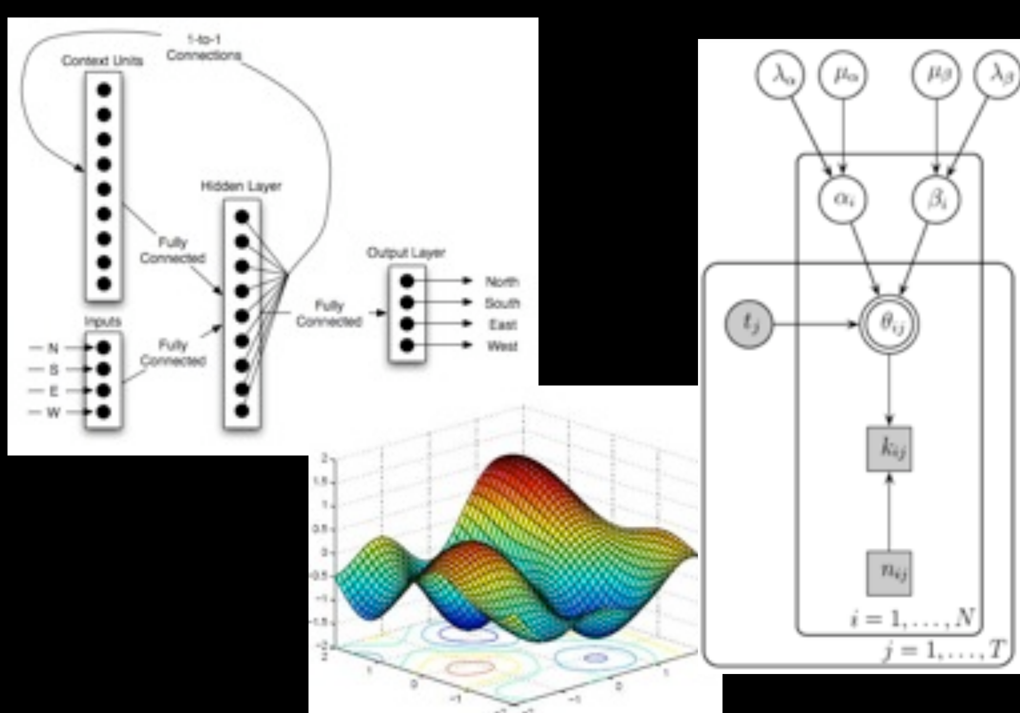


Probabilistic inference

Thought is productive: "the infinite use of finite means"

Statistics and composition

Thought is useful in an uncertain world



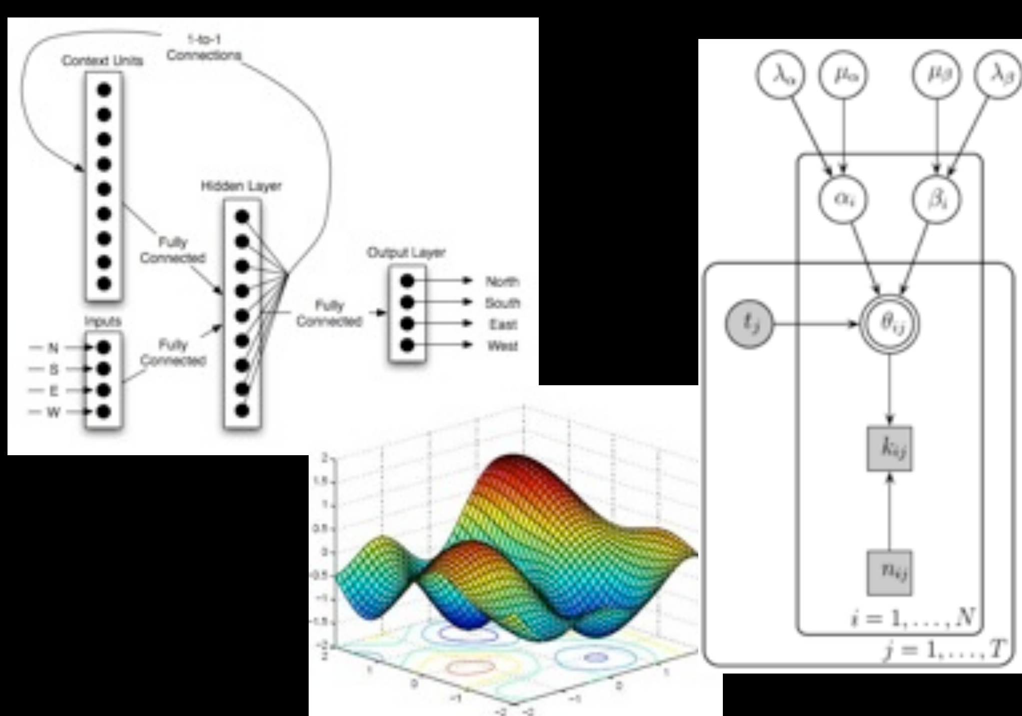
Probabilistic inference

Thought is productive: "the infinite use of finite means"

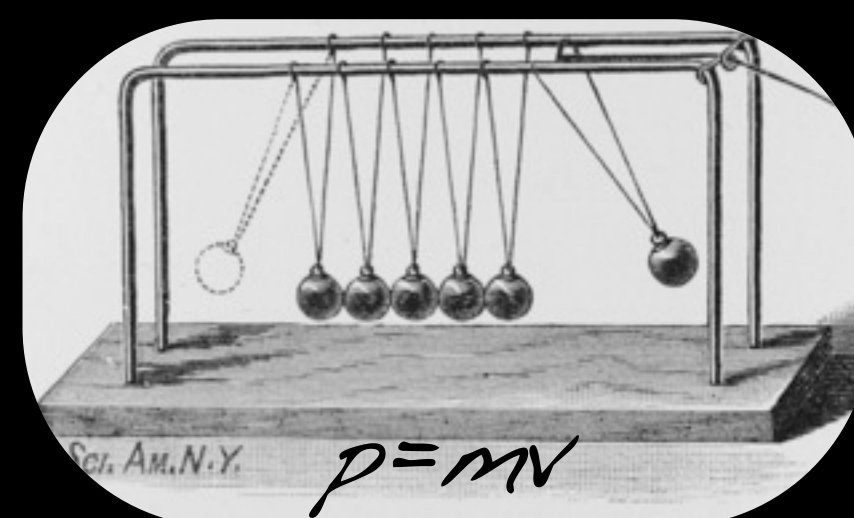
Statistics and composition

Thought is useful in an uncertain world

Thought is productive: "the infinite use of finite means"



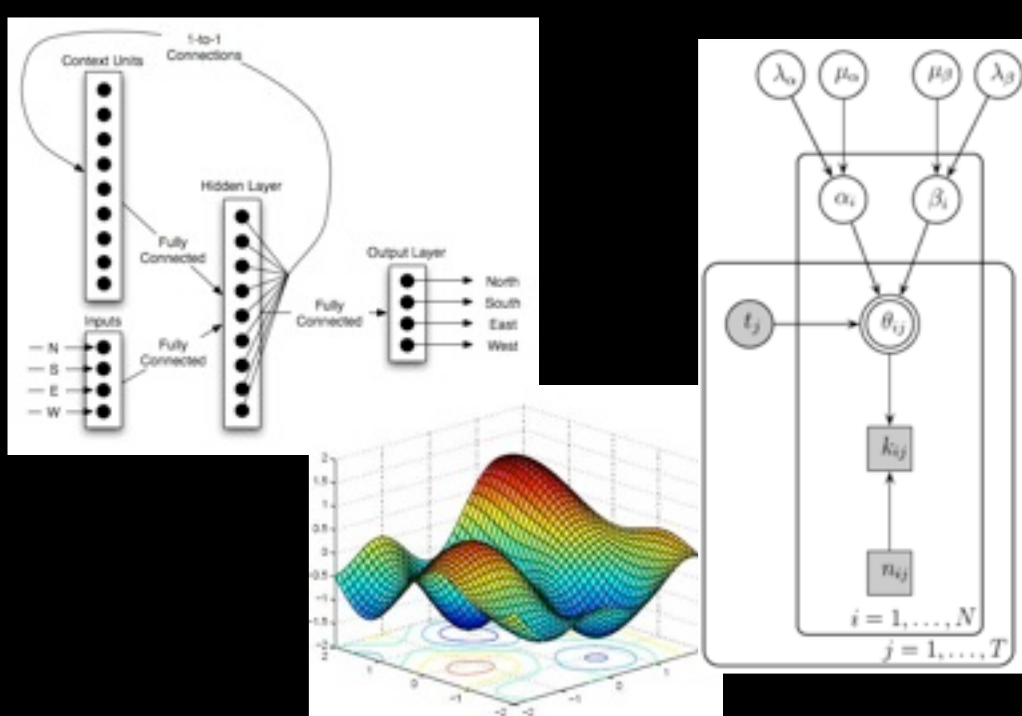
Probabilistic inference



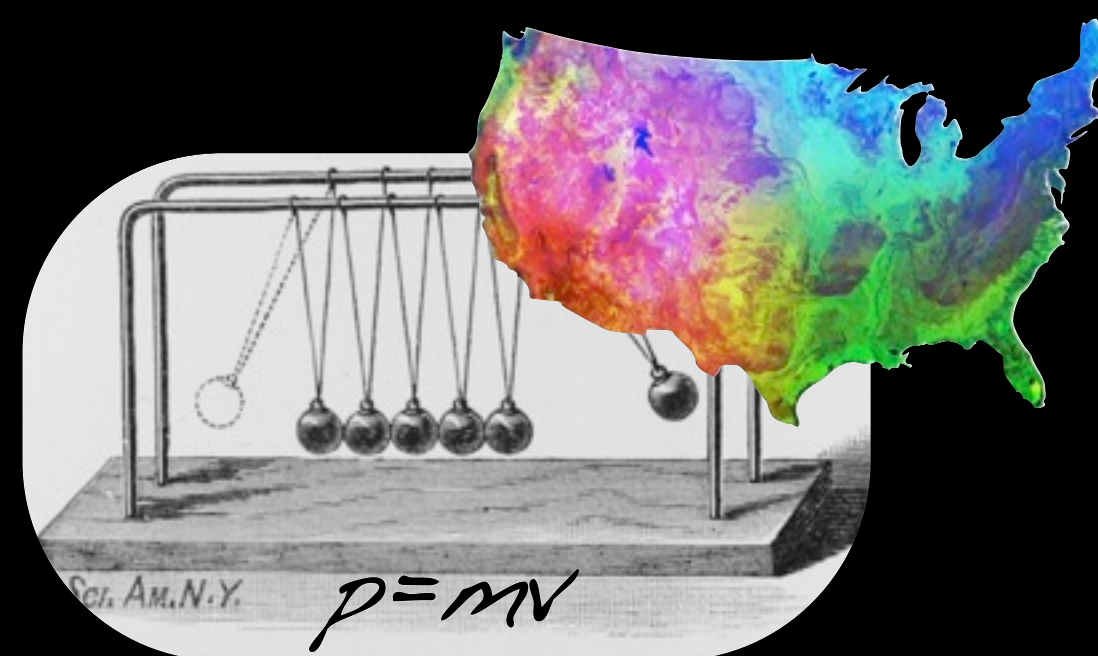
Statistics and composition

Thought is useful in an uncertain world

Thought is productive: "the infinite use of finite means"



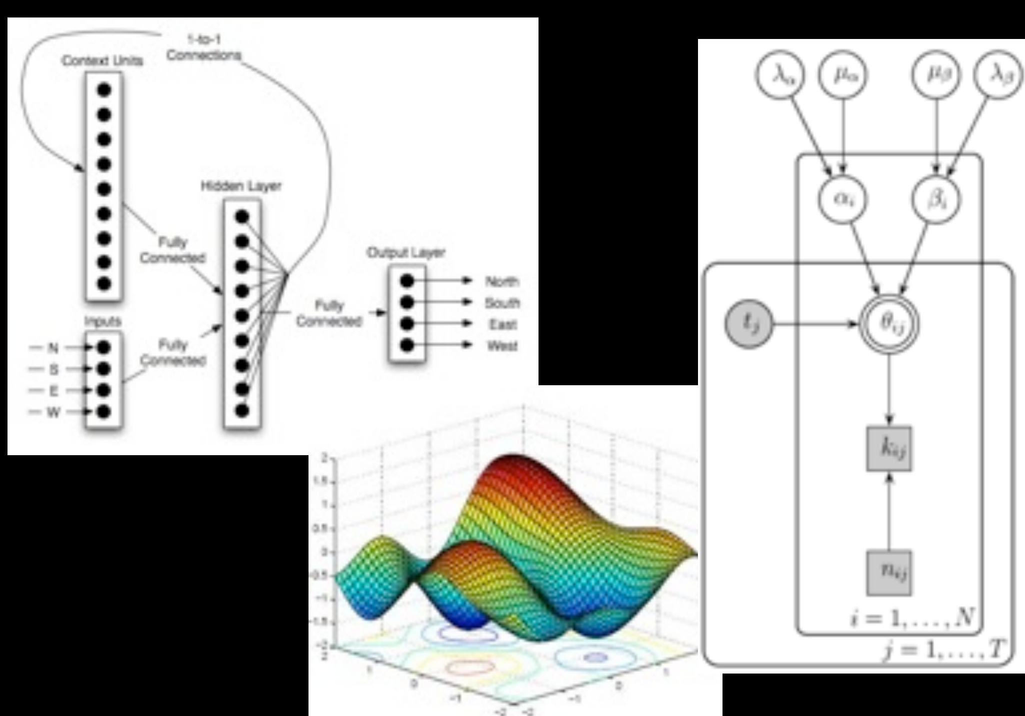
Probabilistic inference



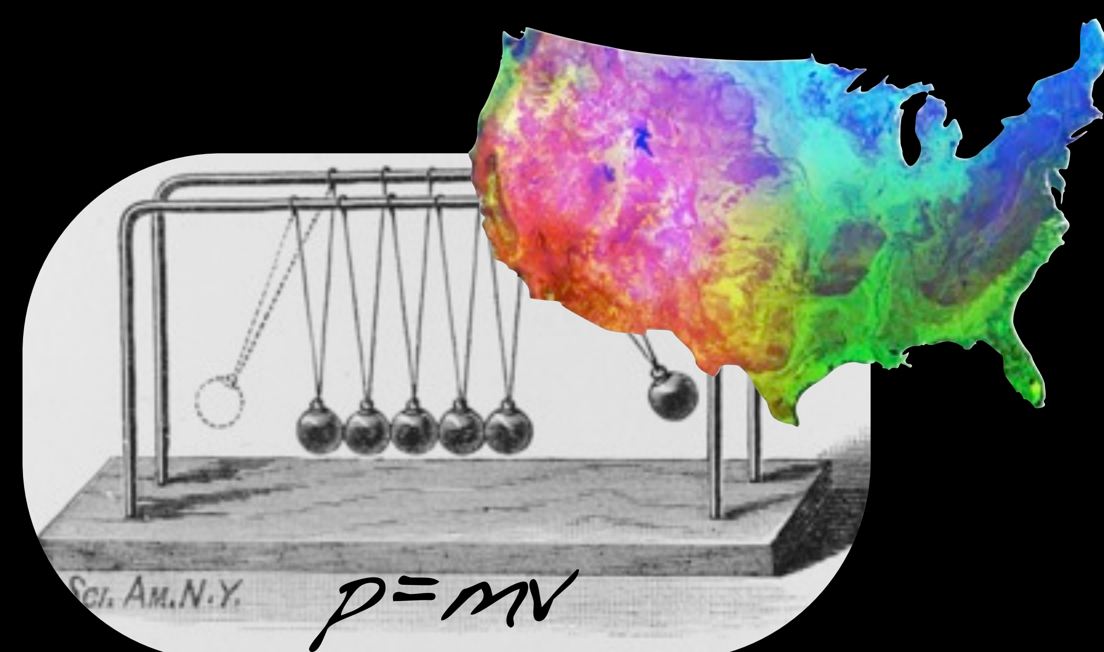
Statistics and composition

Thought is useful in an uncertain world

Thought is productive: "the infinite use of finite means"



Probabilistic inference

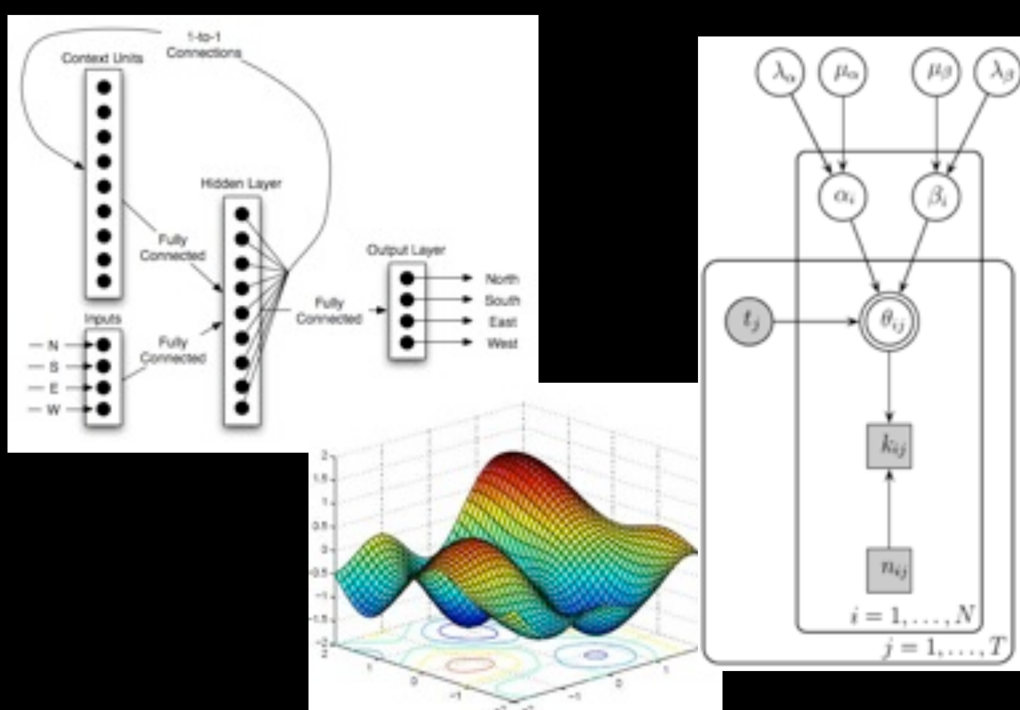


Compositional representations

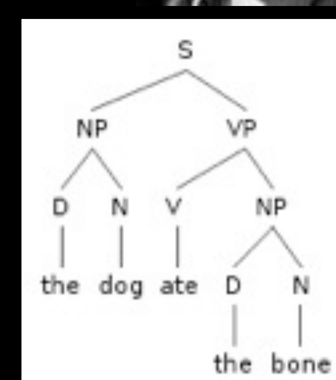
Statistics and composition

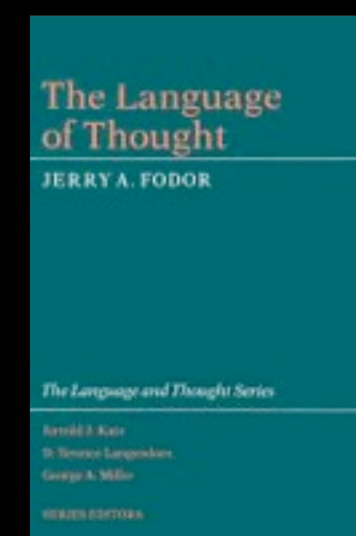
Thought is useful in an uncertain world

Thought is productive: "the infinite use of finite means"



Probabilistic inference





$$\forall x \; King(x) \implies Man(x)$$

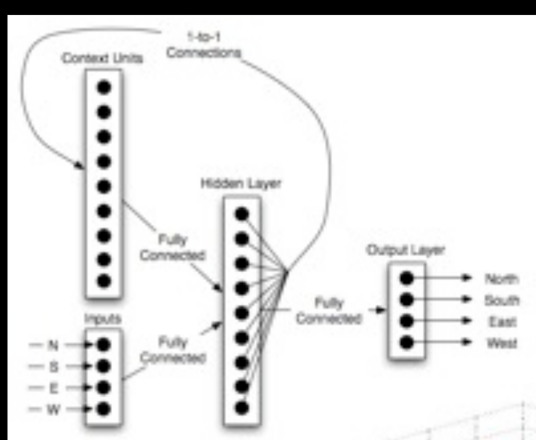
$$\forall y \; Man(y) \iff \neg Woman(y)$$

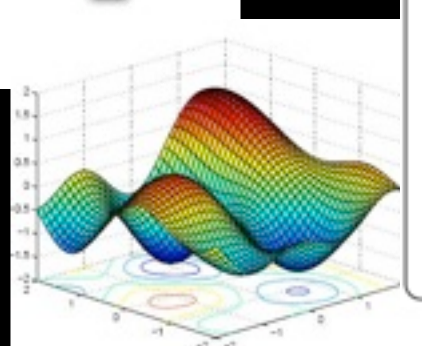
Compositional representations

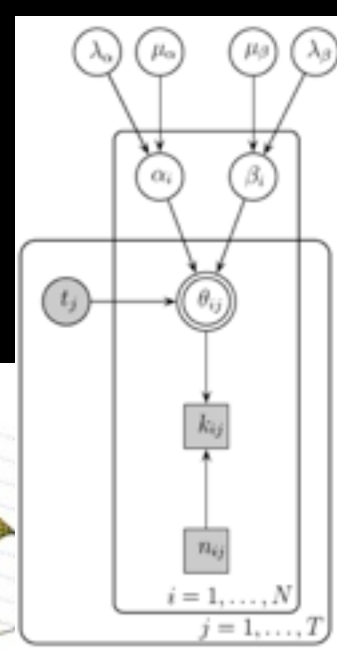
Statistics and composition

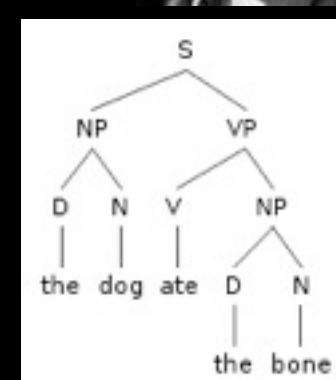
Thought is useful in an uncertain world

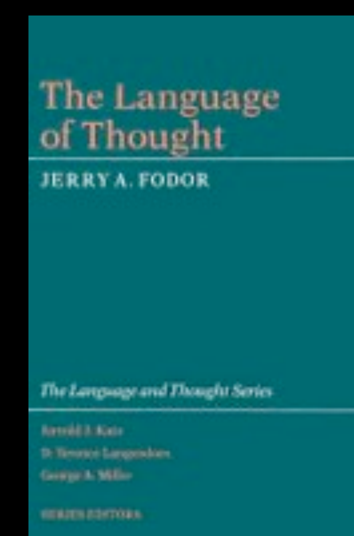
Thought is productive: "the infinite use of finite means"











$$\forall x\ King(x) \implies Man(x)$$

$$\forall y\ Man(y) \iff \neg Woman(y)$$

Probabilistic inference

Generative models

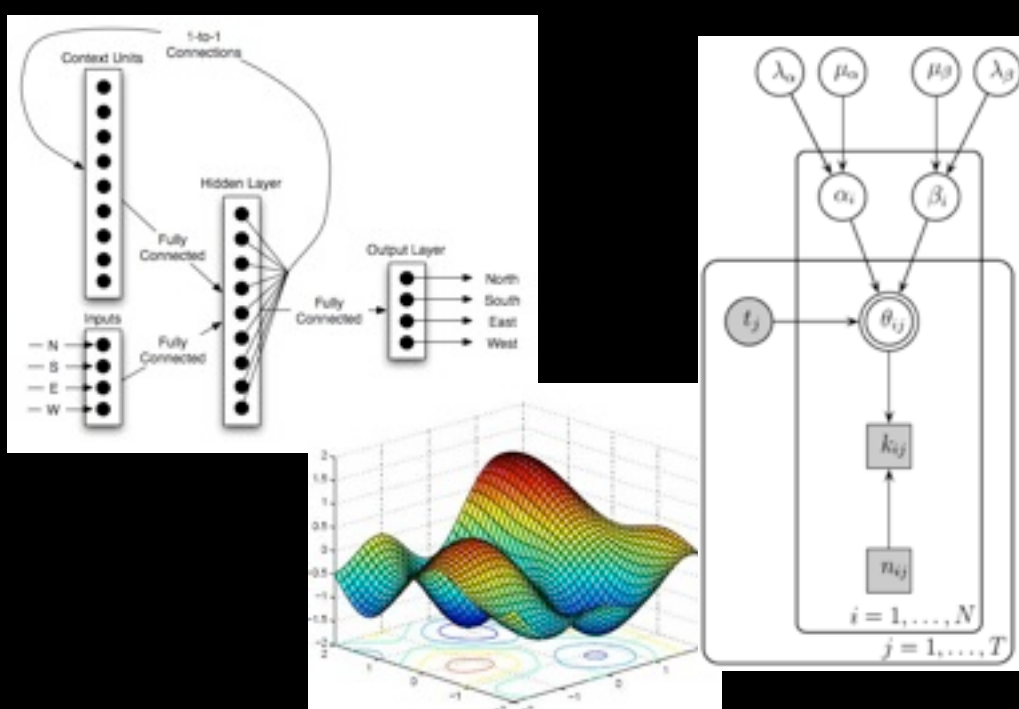
Compositional representations

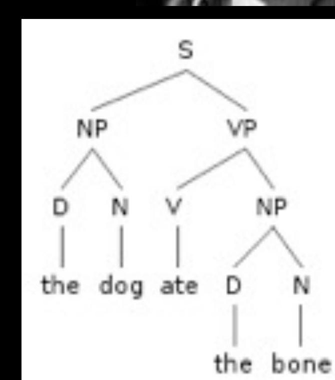
Statistics and composition

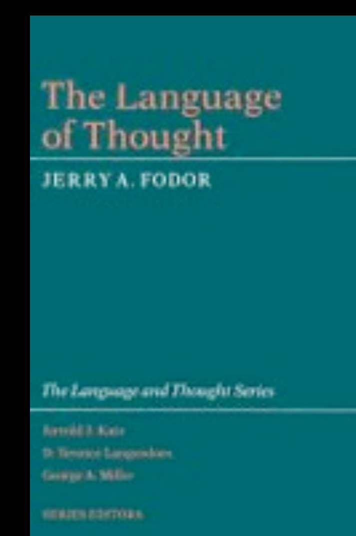
Probabilistic language of thought hypothesis

Thought is useful in an uncertain world

Thought is productive: “the infinite use of finite means”







$$\forall x \; King(x) \implies Man(x)$$

$$\forall y \; Man(y) \iff \neg Woman(y)$$

Probabilistic inference

Generative models

Compositional representations

Probabilistic LoT

- The **probabilistic** language of thought hypothesis:
  - Mental representations are compositional,
  - Their meaning is probabilistic,
  - They encode generative knowledge,
- Hence, they support thinking and learning by probabilistic inference.

Probabilistic LoT

- The **probabilistic** language of thought hypothesis:
  - Mental representations are compositional,
  - Their meaning is probabilistic,
  - They encode generative knowledge,
- Hence, they support thinking and learning by probabilistic inference.

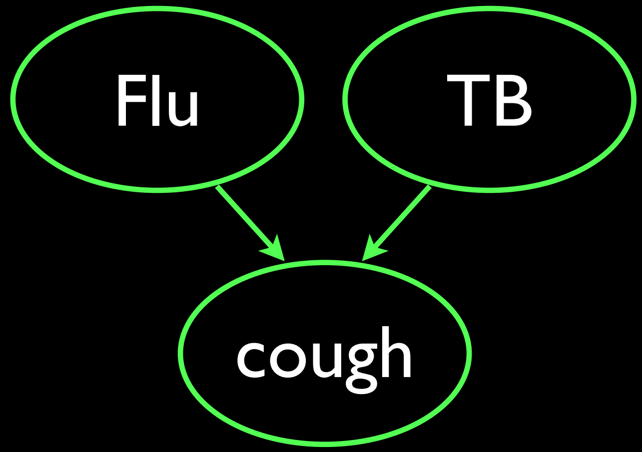
Can this hypothesis be formalized?

Probabilistic generative models

- Mental model of the causal process that gives rise to observations.

Probabilistic generative models

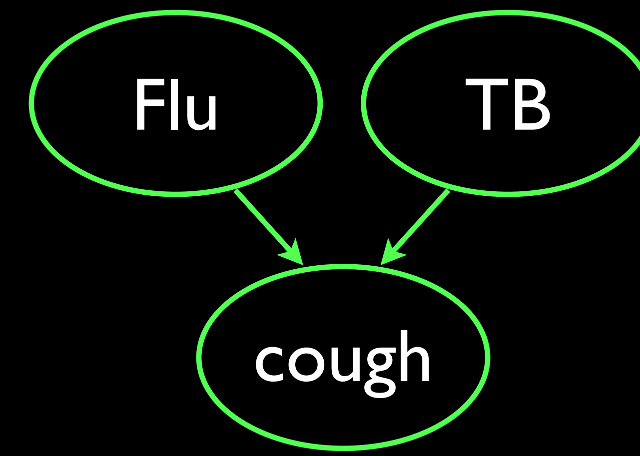
- Mental model of the causal process that gives rise to observations.
- E.g. Bayes nets.



cough	TB	no TB
flu	0.8	0.7
no flu	0.7	0.1

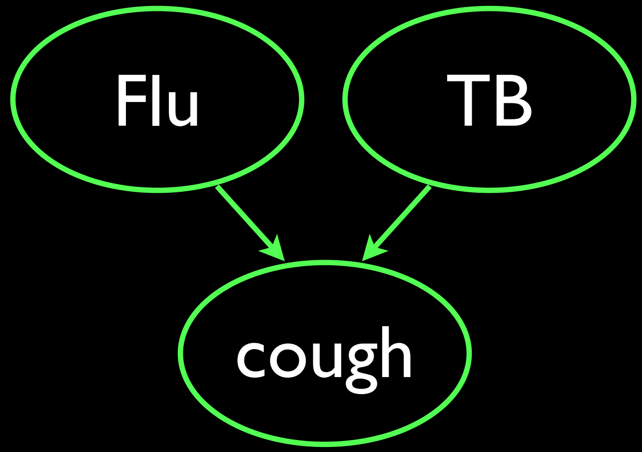
Probabilistic generative models

- Mental model of the causal process that gives rise to observations.
- E.g. Bayes nets.
  - But what's "hidden inside" the nodes and arrows?



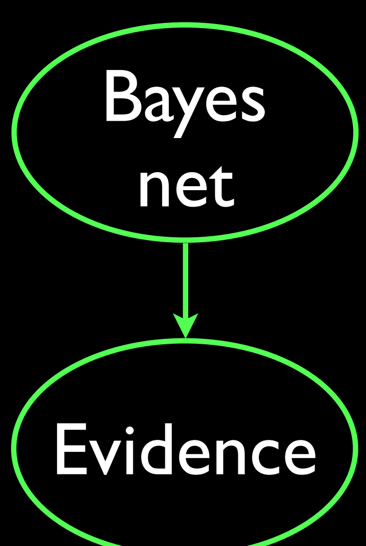
cough	TB	no TB
flu	0.8	0.7
no flu	0.7	0.1

Probabilistic generative models

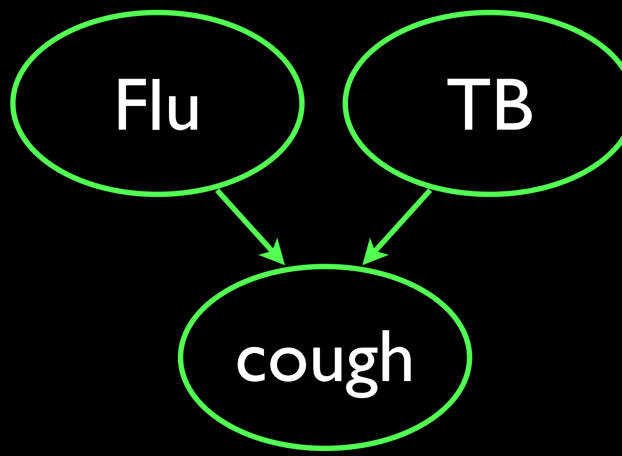


cough	TB	no TB
flu	0.8	0.7
no flu	0.7	0.1

- Mental model of the causal process that gives rise to observations.
- E.g. Bayes nets.
  - But what's "hidden inside" the nodes and arrows?

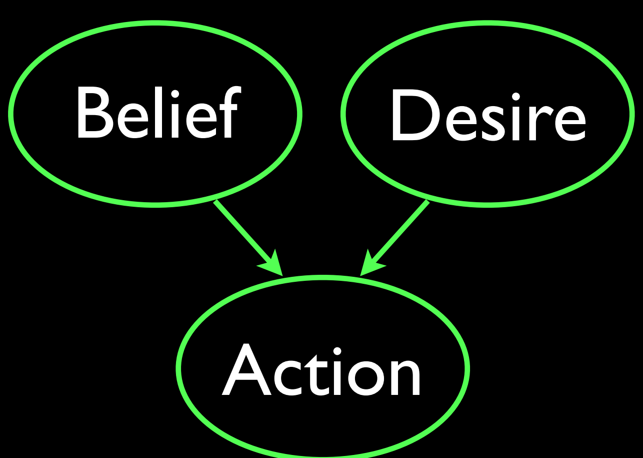


Probabilistic generative models

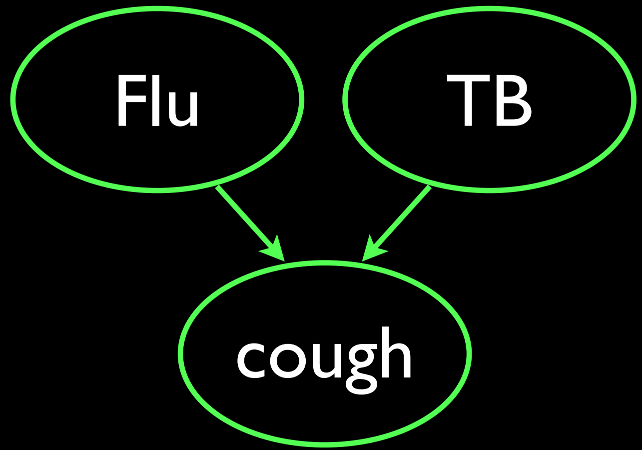


cough	TB	no TB
flu	0.8	0.7
no flu	0.7	0.1

- Mental model of the causal process that gives rise to observations.
- E.g. Bayes nets.
  - But what's "hidden inside" the nodes and arrows?

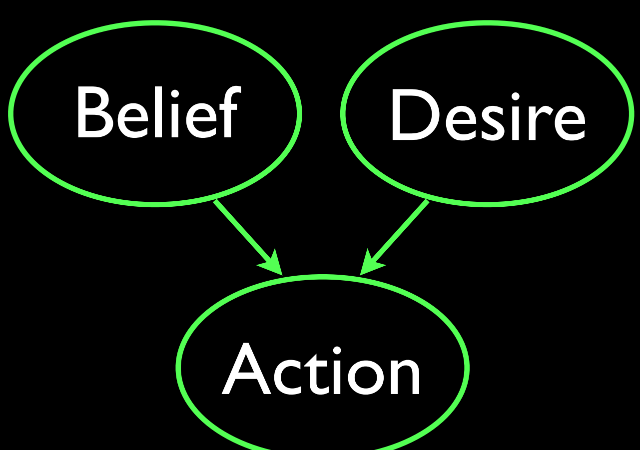


Probabilistic generative models



- Mental model of the causal process that gives rise to observations.
- E.g. Bayes nets.
  - But what's "hidden inside" the nodes and arrows?

cough	TB	no TB
flu	0.8	0.7
no flu	0.7	0.1



write talk	want fame	want sleep	..
going to IPAM	0.7	0.05	
it's raining	0.002	0.004	
:			

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)))))
```

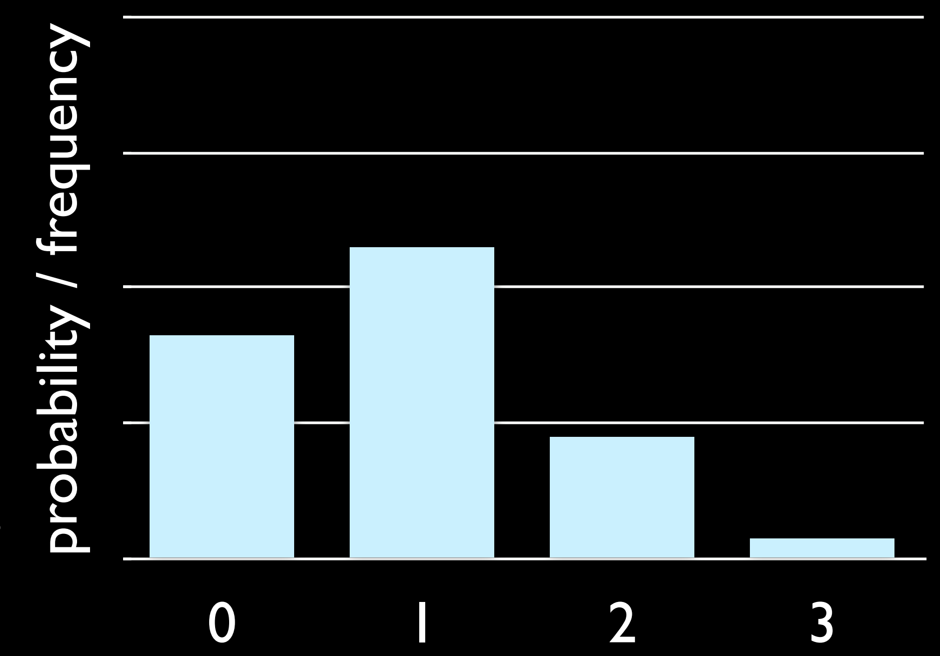
`((repeat double) 3)` => 12

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

Church

Random primitives:

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



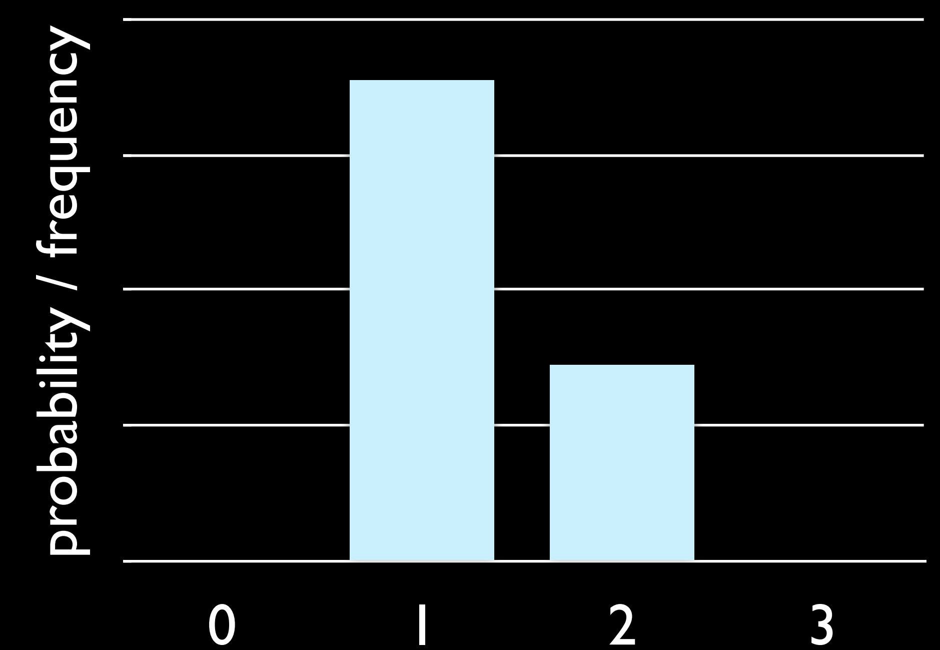
Conditioning (inference):

```
(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)

PLoT

http://projects.csail.mit.edu/church

PLoT

- Formalizing the PLoT:
  Mental representations (concepts) are functions in a **stochastic** lambda calculus (e.g. Church).

http://projects.csail.mit.edu/church

PLoT

- Formalizing the PLoT:
  Mental representations (concepts) are functions in a **stochastic** lambda calculus (e.g. Church).
  - ...or pieces of probabilistic programs, anyhow.

http://projects.csail.mit.edu/church

PLoT

- Formalizing the PLoT:
  Mental representations (concepts) are functions in a **stochastic** lambda calculus (e.g. Church).
  - ...or pieces of probabilistic programs, anyhow.
  - Separate the process of inference from representations and the inferences they license (Cf. Marr's levels).

http://projects.csail.mit.edu/church

Outline (next 3 lectures)

- Church: language design considerations.
- Church: inference techniques.
- Social cognition.
- Natural language pragmatics (etc).
- Concept learning as program induction.

Outline (next 3 lectures)

- Church: language design considerations.
- Church: inference techniques.
- Social cognition.
- Natural language pragmatics (etc).
- Concept learning as program induction.

Church language design

- Start with (pure subset of) LISP.
- Add random primitives.
- Add `mem`.
- Add `query`.

Why functions?

- Composition of probabilistic functions represents *directed* generative models.
  - Captures the intuition that much human knowledge is about causal process.
    - For example, explaining away: conditioning an undirected model can introduce new independence but not new dependence.
  - Hierarchical models, etc.

Undirected modeling?

- Complex conditions in `query` represent (any) undirected constraints.
- For example an Ising model:

```
(query
 (define a (flip))
 (define b (flip))
 (define c (flip))

 (list a b c)

 (and (flip (if (equal? a b)) 1.0 0.3)
      (flip (if (equal? b c)) 1.0 0.3)))
```

Undirected modeling?

- Complex conditions in `query` represent (any) undirected constraints.
- For example an Ising model:

```
(query
 (define a (flip))
 (define b (flip))
 (define c (flip))

 (list a b c)

 (and (flip (if (equal? a b)) 1.0 0.3)
      (flip (if (equal? b c)) 1.0 0.3)))
```

$e^{\ln(0.3)\delta_{a=b}}$

Undirected modeling?

- Complex conditions in `query` represent (any) undirected constraints.
- For example Markov Logic:

```
(query
  (define world (repeat N flip))
  world
  (and
   (log-flip (if (clause1 world) 0.0 weight1)))
   (log-flip (if (clause2 world) 0.0 weight2)))
   ...))
```

Undirected modeling?

- Complex conditions in `query` represent (any) undirected constraints.
- So, Church can represent directed, undirected, and hybrid models.
  - Complex conditions can be built out of directed pieces.
    (This seems important for natural language. Cf. Montegue.)

Why functional?

- First class abstraction.
  - The pieces needed for model specification are representable in the modeling language.
  - Abstraction permits domain specific languages -- changes the *effective* language of thought.
  - Higher-order functions are useful.
    (E.g. the higher-order distribution DPmem.)
  - Recursion.
- Models are parsimonious.

Why LISP?

- Small core language.
  - Simple semantics.
  - Very little syntax.
  - Simple type system.
- Code as data (meta-circular, homoiconic).
- Well-worked out concepts for computation (see SICP).

Random primitives

- Just the fair-coin `flip` primitive is enough (ψλ-calculus),
- But it is convenient (and well-formed) to add other random primitives.
- An ERP is a primitive (with arguments) that can sample values and score.
  - For each set of arguments, multiple samples from an ERP must be independent identically distributed (iid)
  - Examples: `flip`, `sample-discrete`, `gaussian`.

Exchangeability

- If all the ERPs return iid samples, then is any Church thunk (procedure without arguments) an iid distribution?
- No. Consider:
- But they are *exchangeable*.

```
(define weight (beta 1 1))
(define (mycoin) (flip weight))
(repeat 10 mycoin)
```

```
=> (t t t t t t t t t t)
```

- Theorem (Freer & Roy, 2010): Any church thunk is exchangeable and any exchangeable distribution may be represented by a Church thunk.

Random primitives

- Relax the interface for random primitives: an XRP can sample and score. Multiple calls *exchangeable*.
- Examples:
  - Uniform draw without replacement.
  - Gensym.
  - Beta-binomial.
  - CRP.

Memoization

```
(= (flip) (flip))
```
⟶ True with probability 0.5

```
(define mem-flip (mem flip))
(= (mem-flip) (mem-flip))
```
⟶ True with probability 1.0

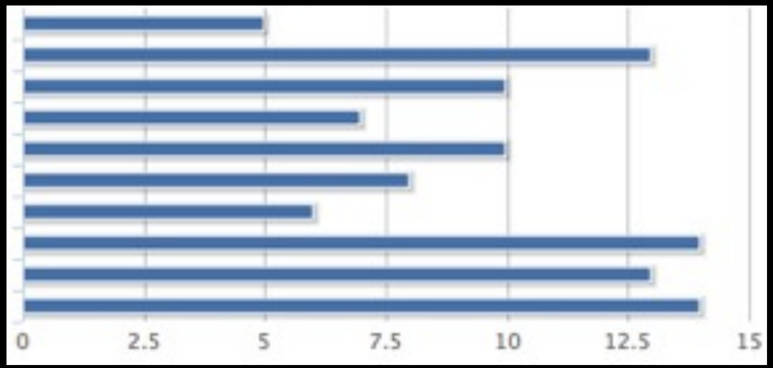
- The mem primitive memoizes a procedure.
  - This changes the semantics (unlike in deterministic languages).
  - Interacts with symbols and `gensym`, to enable "BLOG style" style world models.

```
(define people (repeat (poisson 1.0) gensym))
(define eye-color
   (mem (λ (person) (uniform-draw '(blue brown)))))
(define blue-eyed-people
 (filter (λ (person) (equal? 'blue (eye-color person)))
         people))
```

Stochastic memoization

- `DPmem`: stochastically reuse return values from functions.
  - A *higher-order* distribution: it takes a stochastic function (of any type) and samples a new function (of same type) that concentrates the original.
  - Adapts a generative process to balance reuse with re-computation (see Johnson, O'Donnell).
  - Implemented in Church via stick-breaking or using an XRP.

```
(define mem-flip (DPmem 1.0 flip))
(= (mem-flip) (mem-flip))
```



DP mixture model

```
(define draw-class
   (DPmem 1.0 gensym))
(define class
   (mem (lambda (obj) (draw-class))))
(define class-weight
   (mem (lambda (obj-class feature)
             (beta 1.0 1.0))))
(define observe-feature
   (lambda (obj feature)
     (flip (class-weight (class obj) feature))))
```

DP mixture model

```
(define draw-class
   (DPmem 1.0 gensym))
(define class
   (mem (lambda (obj) (draw-class))))
(define class-weight
   (mem (lambda (obj-class feature)
               (beta 1.0 1.0))))
(define observe-feature
   (lambda (obj feature)
     (flip (class-weight (class obj) feature))))
```

A pool of cluster symbols with DP prior.

The cluster of an object and cluster parameters are persistent.

Infinite relational model

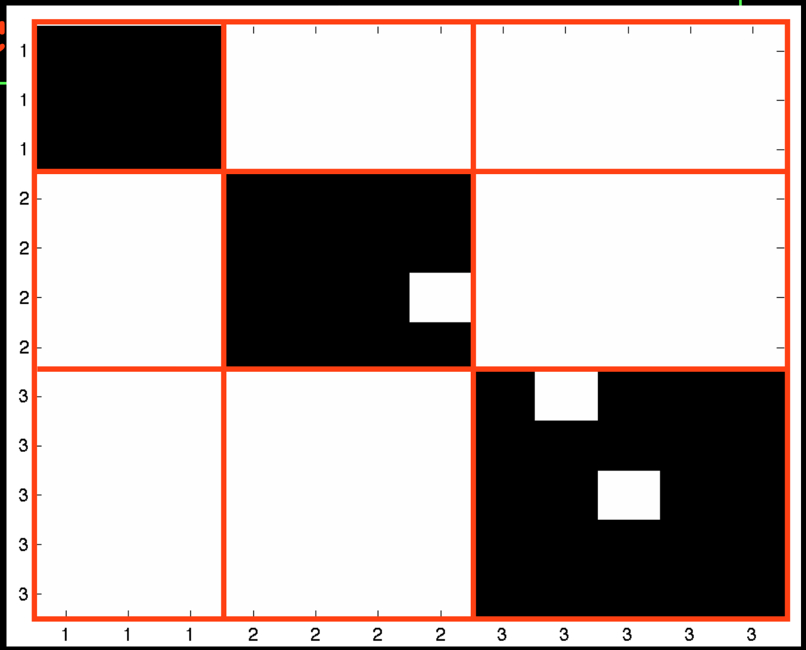
The IRM model from: Kemp, et al, 2006

```
(define draw-class
   (DPmem 1.0 (lambda (obj-domain) (gensym))))
(define class
   (mem (lambda (obj) (draw-class (domain obj)))))
(define class-weight
   (mem (lambda (obj-class1 obj-class2)
                (beta 1.0 1.0))))
(define observe-relation
   (lambda (obj1 obj2)
     (flip (class-weight (class obj1) (class obj2)))))
```

Infinite relational model

The IRM model from: Kemp, et al, 2006

```
(define draw-class
   (DPmem 1.0 (lambda (obj-domain) (gensym))))
(define class
   (mem (lambda (obj) (draw-class (domain obj)))))
(define class-weight
   (mem (lambda (obj-class1 obj-class2)
                (beta 1.0 1.0))))
(define observe-relation
   (lambda (obj1 obj2)
     (flip (class-weight (class obj1) (c
```



Nested CRP

```
(define category
  (DPmem 1.0
         (λ (parent-category)
           (pair (gensym) parent-category))))

(define (category-hierarchy N)
  (if (= N 0)
      '(top)
      (category (category-hierarchy (- N 1)))))

(category-hierarchy 3) x4...
```

Nested CRP

```
(define category
  (DPmem 1.0
         (λ (parent-category)
           (pair (gensym) parent-category))))

(define (category-hierarchy N)
  (if (= N 0)
      '(top)
      (category (category-hierarchy (- N 1)))))

(category-hierarchy 3) x4...
```

```
=> ((g0 g1 g2 top)
    (g3 g1 g2 top)
    (g4 g1 g2 top)
    (g5 g6 g7 top))
```

HDP-HMM

```
(define draw-state (DPmem 1.0 gensym))
(define transition (DPmem 1.0 (λ (state) (draw-state))))
(define obsfn
  (mem (λ (state) (make-dirichlet-multi terminals)))

(define (hdp-hmm state N)
 (if (= N 0)
     ‘()
     (pair ((obsfn state))
           (hdp-hmm (transition state) (- N 1)))))
```

first-class `query`

- How should we specify observations in probabilistic programs?
  - Separate model from data / data variables?
  - ‘Observe’ statements within model?
  - Better: make inference an ordinary function.
    - Can define inference *within* the language (e.g. by rejection).
    - Gives proper scoping, complex conditions, nested query (‘inference about inference’).

Nested query

Alice and Bob arrange to meet at the bar, each later realizes they didn't fix which bar and must guess where to meet.

(A coordination game, see Schelling, 1960)

Recursive social inference:

```
(define (sample-location)
  (if (flip .55) 'popular-bar 'unpopular-bar))

(define (bob depth)
  (query
    (define bob-location (sample-location))
    bob-location
    (equal? bob-location (alice (- depth 1)))))

(define (alice depth)
  (query
    (define alice-location (sample-location))
    alice-location
    (or (= depth 0)
        (equal? alice-location (bob depth)))))
```

Nested query

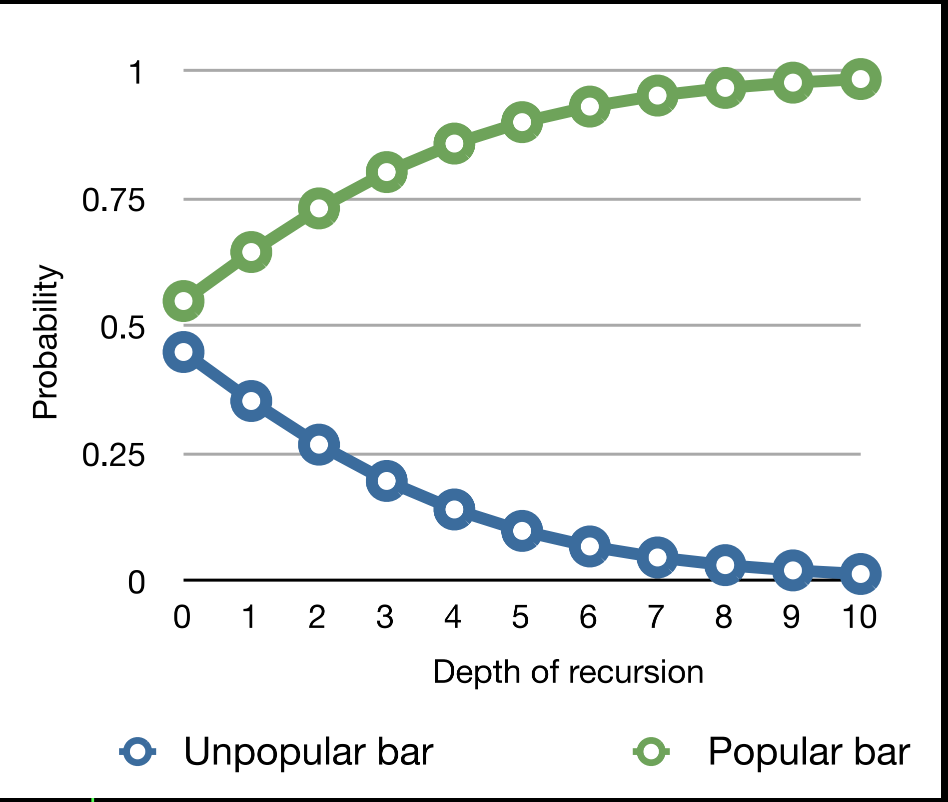
Alice and Bob arrange to meet at the bar, each later realizes they didn't fix which bar and must guess where to meet.
(A coordination game, see Schelling, 1960)

Recursive social inference:

```
(define (sample-location)
  (if (flip .55) 'popular-bar 'unpopular-bar))

(define (bob depth)
  (query
    (define bob-location (sample-location))
    bob-location
    (equal? bob-location (alice (- depth 1)))))

(define (alice depth)
  (query
    (define alice-location (sample-location))
    alice-location
    (or (= depth 0)
        (equal? alice-location (bob depth)))))
```



Nested query

Policy-free MDP planning as a
recursively optimal action selection.

```
(define (choose-action state)
  (query
   (define action (action-prior))
   action
   (flip (normalize-reward
           (sample-reward action state)))))

(define (sample-reward action state)
  (let ((next-state (state-transition state action)))
      (+ (reward next-state)
         (if (terminal? next-state)
             0
             (sample-reward
              (choose-action next-state)
              next-state)))))
```

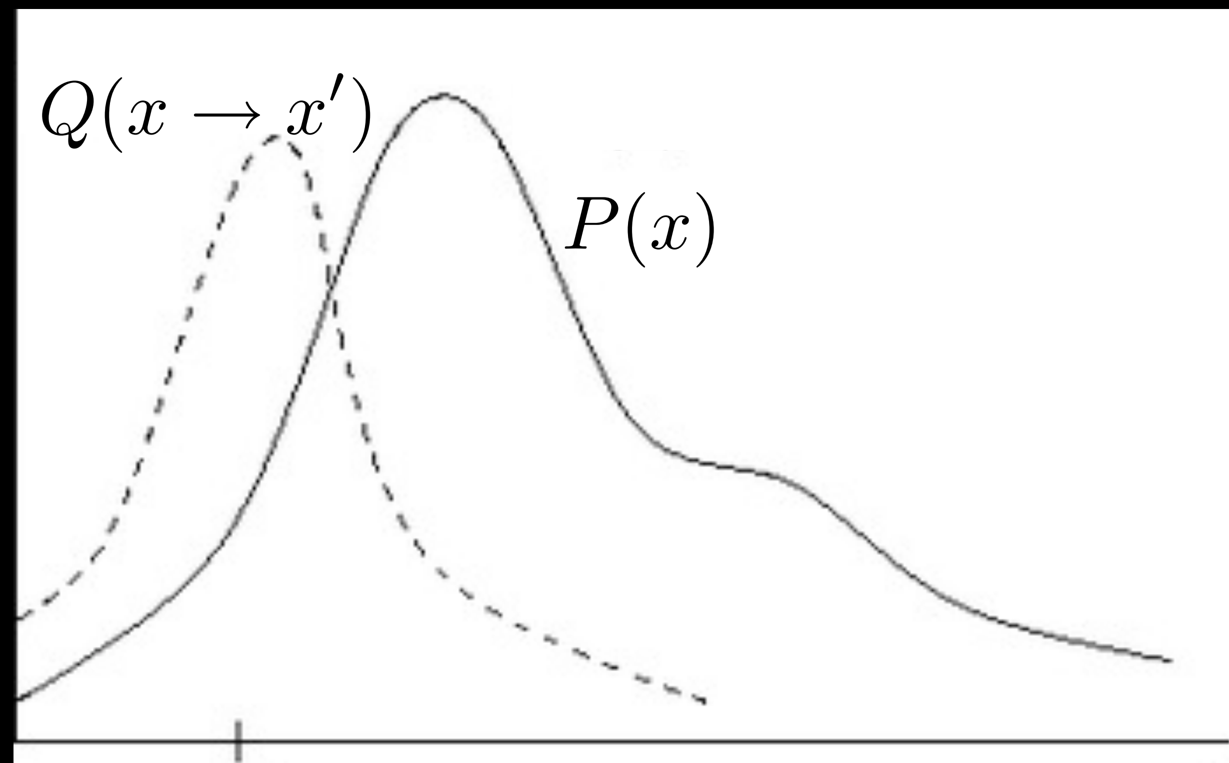
Church vs. ...

- Bugs: Less expressive language for directed models. More stable engine.
- BLOG: Nice constructs for unknown objects. No abstraction.
- IBAL (etc): Similar. No continuous variables, mem, xrps. Obs not query.
- MLN: Undirected, based on FoL. Good implementations.
- Csoft, Factorie, Figaro, Hansei, ProbLog....

Church inference

- Universal representation (PLoT) needs universal inference.
  - The mind can't have a separate algorithm for every task.
- Rejection sampling is sound, but slow.
- Efficient universal inference?

Metropolis-Hastings



- To sample from $P(x)$, repeatedly:
  - propose x' from $Q(x \to x')$,
  - accept with probability: $\min\left(1, \frac{P(x)Q(x \to x')}{P(x')Q(x' \to x)}\right)$
- This converges in distribution to $P(x)$ (even if $P(x)$ isn't normalized).

MH for Church

- What is a state?
  - The set of random choices made in executing (forward) the query expression.
- How can choices be individuated?
  - Not by evaluation order (can change)...

MH for Church

- What is a state?
  - The set of random choices made in executing (forward) the query expression.
- How can choices be individuated?
  - Not by evaluation order (can change)...

```
(and
  (if (flip)
      (flip)
      true)
  (flip))
```

MH for Church

- What is a state?
  - The set of random choices made in executing (forward) the query expression.
- How can choices be individuated?
  - Not by evaluation order (can change)...

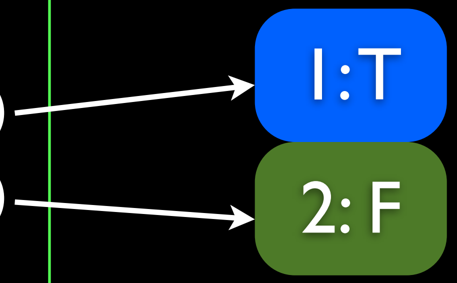
```
(and
  (if (flip)
      (flip)
      true)
  (flip))
```

1:T

MH for Church

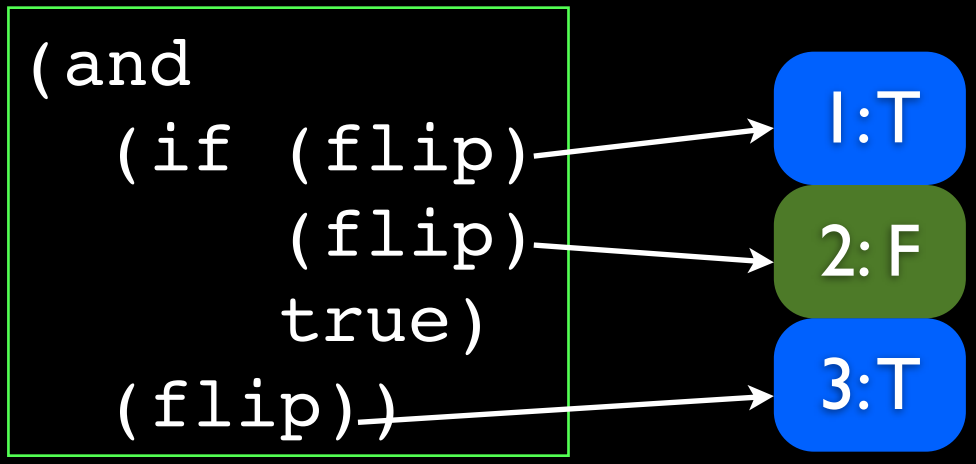
- What is a state?
  - The set of random choices made in executing (forward) the query expression.
- How can choices be individuated?
  - Not by evaluation order (can change)...

```
(and
  (if (flip)
      (flip)
      true)
  (flip))
```



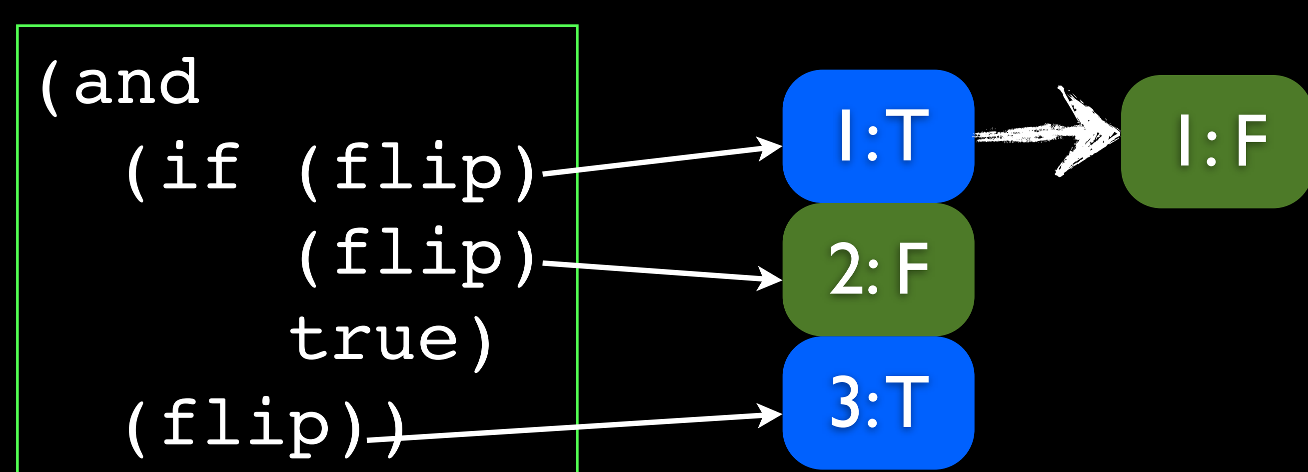
MH for Church

- What is a state?
  - The set of random choices made in executing (forward) the query expression.
- How can choices be individuated?
  - Not by evaluation order (can change)...



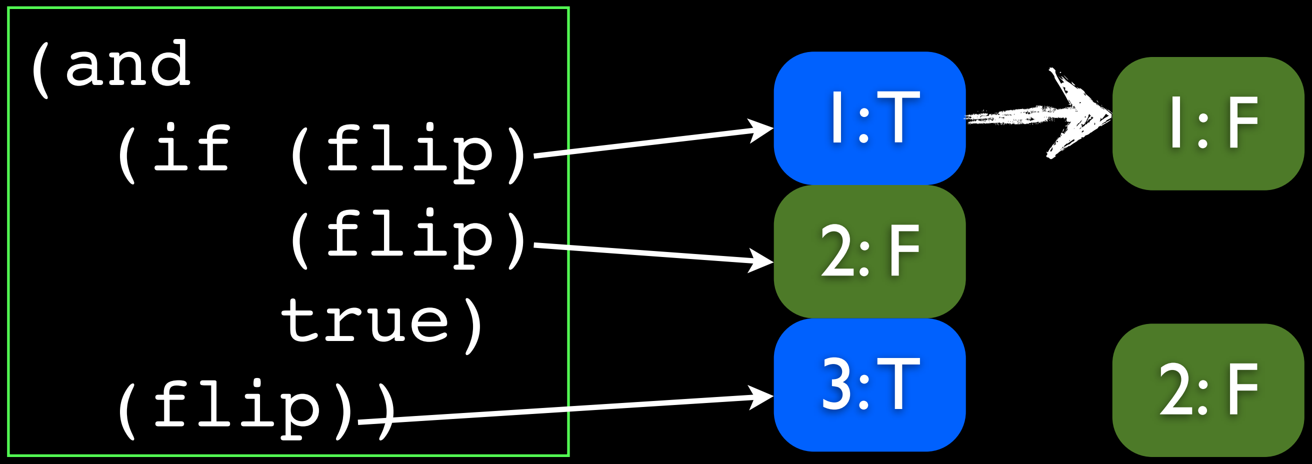
MH for Church

- What is a state?
  - The set of random choices made in executing (forward) the query expression.
- How can choices be individuated?
  - Not by evaluation order (can change)...



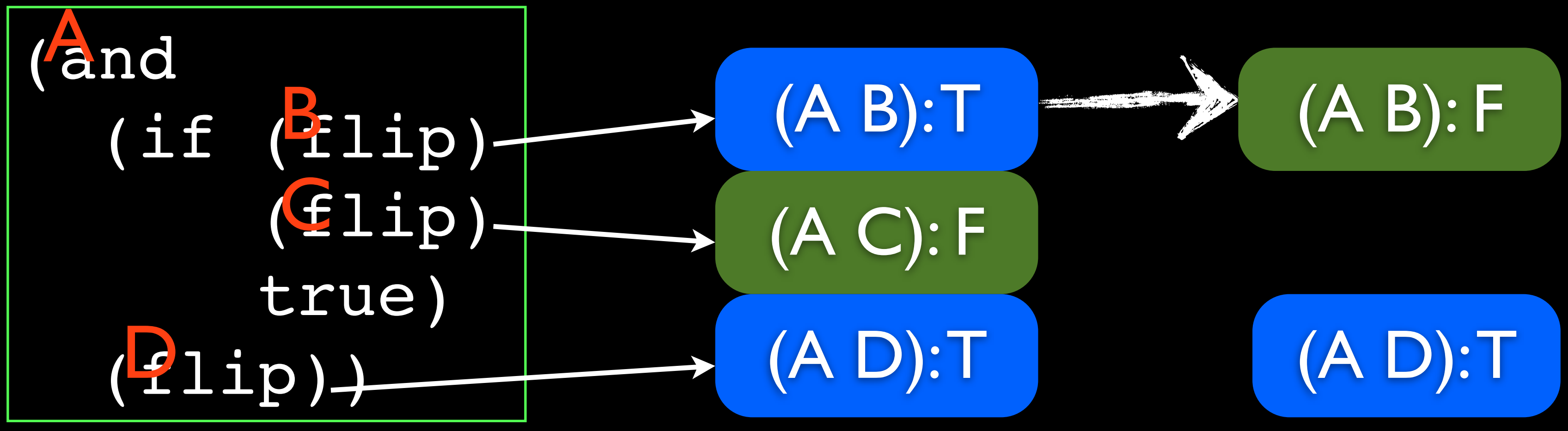
MH for Church

- What is a state?
  - The set of random choices made in executing (forward) the query expression.
- How can choices be individuated?
  - Not by evaluation order (can change)...



MH for Church

- What is a state?
  - The set of random choices made in executing (forward) the query expression.
- How can choices be individuated?
  - Not by evaluation order (can change)...
  - Named via dynamic calling address:



MH for Church

- What is a state?
  - The set of random choices made in executing (forward) the query expression.
- How can choices be individuated?
  - Not by evaluation order (can change)...
  - Named via dynamic calling address:

$\mathcal{A}^{top}[\![E]\!] = $ `((lambda (addr)` $\mathcal{A}[\![E]\!]$`) '(top))`

$\mathcal{A}[\![$`(lambda (`$I_{i=1}^{n}$`)` E_{body}`)`$]\!] = $ `(lambda (addr .` $I_{i=1}^{n}$`)` $\mathcal{A}[\![E_{body}]\!]$`)`

where $\mathcal{S}$ is a globally unique symbol.

$\mathcal{A}[\![$`(mem` E`)`$]\!] = $ `((lambda (maddr f) (lambda (addr . args) (apply f (cons args maddr) args))) addr` $\mathcal{A}[\![E]\!]$`)`

$\mathcal{A}[\![$`(begin` $E_{i=1}^{n}$`)`$]\!] = $ `(begin` $\mathcal{A}[\![E_i]\!]_{i=1}^{n}$`)`

$\mathcal{A}[\![$`(letrec ((`I_i E_i`)`$_{i=1}^{n}$`)` E_{body}`)`$]\!] = $ `(letrec ((`I_i $\mathcal{A}[\![E_i]\!]$`)`$_{i=1}^{n}$`)` $\mathcal{A}[\![E_{body}]\!]$`)`

$\mathcal{A}[\![$`(if` E_t E_c E_a`)`$]\!] = $ `(if` $\mathcal{A}[\![E_t]\!]$ $\mathcal{A}[\![E_c]\!]$ $\mathcal{A}[\![E_a]\!]$`)`

$\mathcal{A}[\![$`(define` I E`)`$]\!] = $ `(define` I $\mathcal{A}[\![E]\!]$`)`

$\mathcal{A}[\![$`(quote` E`)`$]\!] = $ `(quote` E`)`

$\mathcal{A}[\![$`(`E_{op} $E_{i=1}^{n}$`)`$]\!] = $ `(`$\mathcal{A}[\![E_{op}]\!]$ `(cons '`$\mathcal{S}$ `addr)` $\mathcal{A}[\![E_i]\!]_{i=1}^{n}$`)`

$\mathcal{A}[\![E]\!] = E$, otherwise.

MH for Church

- What is a state?
  - The set of random choices made in executing (forward) the query expression.
- How can choices be individuated?
  - Not by evaluation order (can change)...
  - Named via dynamic calling address:

$\mathcal{A}^{top}[\![E]\!] = \texttt{((lambda (addr) } \mathcal{A}[\![E]\!]\texttt{) '(top))}$

$\mathcal{A}[\![\texttt{(lambda (}I_{i=1}^{n}\texttt{) }E_{body}\texttt{)}]\!] = \texttt{(lambda (addr . }I_{i=1}^{n}$

where $\mathcal{S}$ is a globally unique symbol.

$\mathcal{A}[\![\texttt{(mem }E\texttt{)}]\!] = \texttt{((lambda (maddr f) (lambda (add}$

$\mathcal{A}[\![\texttt{(begin }E_{i=1}^{n}\texttt{)}]\!] = \texttt{(begin }\mathcal{A}[\![E_i]\!]_{i=1}^{n}\texttt{)}$

$\mathcal{A}[\![\texttt{(letrec ((}I_i\ E_i\texttt{)}_{i=1}^{n}\texttt{) }E_{body}\texttt{)}]\!] = \texttt{(letrec ((}I_i$

$\mathcal{A}[\![\texttt{(if }E_t\ E_c\ E_a\texttt{)}]\!] = \texttt{(if }\mathcal{A}[\![E_t]\!]\ \mathcal{A}[\![E_c]\!]\ \mathcal{A}[\![E_a]\!]\texttt{)}$

$\mathcal{A}[\![\texttt{(define }I\ E\texttt{)}]\!] = \texttt{(define }I\ \mathcal{A}[\![E]\!]\texttt{)}$

$\mathcal{A}[\![\texttt{(quote }E\texttt{)}]\!] = \texttt{(quote }E\texttt{)}$

$\mathcal{A}[\![\texttt{(}E_{op}\ E_{i=1}^{n}\texttt{)}]\!] = \texttt{(}\mathcal{A}[\![E_{op}]\!]\texttt{ (cons '}\mathcal{S}\texttt{ addr) }\mathcal{A}[\![E_i]\!]_{i=}^{n}$

$\mathcal{A}[\![E]\!] = E$, otherwise.

```
(begin
  (define geometric
    (lambda (p)
      (if (flip p)
          1
          (+ 1 (geometric p)))))
  (geometric .7))
```

$\longrightarrow$

```
((lambda (addr)
   (begin
     (define geometric
       (lambda (addr p)
         (if (flip (cons 'a1 addr) p)
             1
             (+ (cons 'a2 addr)
                1
                (geometric (cons 'a3 addr) p)))))
     (geometric (cons 'a5 addr) 0.7)))
 '(top))
```

MH for Church

- MH over ‘execution traces’:
  - propose: change a single random choice,
  - trace update: re-execute to update the state, reusing existing choices (new choices from conditional prior),
  - collect, the (now) unused choices,
  - check condition and compute score,
  - accept/reject.
- Initial state must satisfy condition (rejection, annealing, constraint propagation).

MH for Church

- Very similar to BLOG algorithm.
  - Evaluation automatically instantiates minimal "partial world" (but cf. lazy evaluation).
- "Lightweight" technique, addressing by program transformation, works for almost any programming language -- enter Stochastic X.
- Current implementation doesn't scale well (due to software engineering issues).

Goodman, et al. (2008)
Wingate, Stuhlmueller, Goodman (2011)

Hamiltonian Monte Carlo

- Add an auxiliary "momentum" variable for each continuous variable.
- Hamiltonian evolution is reversible and conserves probability (hence satisfies detailed balance).

$$\phi(x) = -\ln(P(x))$$

$$H(x,p) = \phi(x) + mp^2$$

$$\frac{dx}{dt} = \frac{\partial H}{\partial p}, \quad \frac{dp}{dt} = -\frac{\partial H}{\partial q}$$

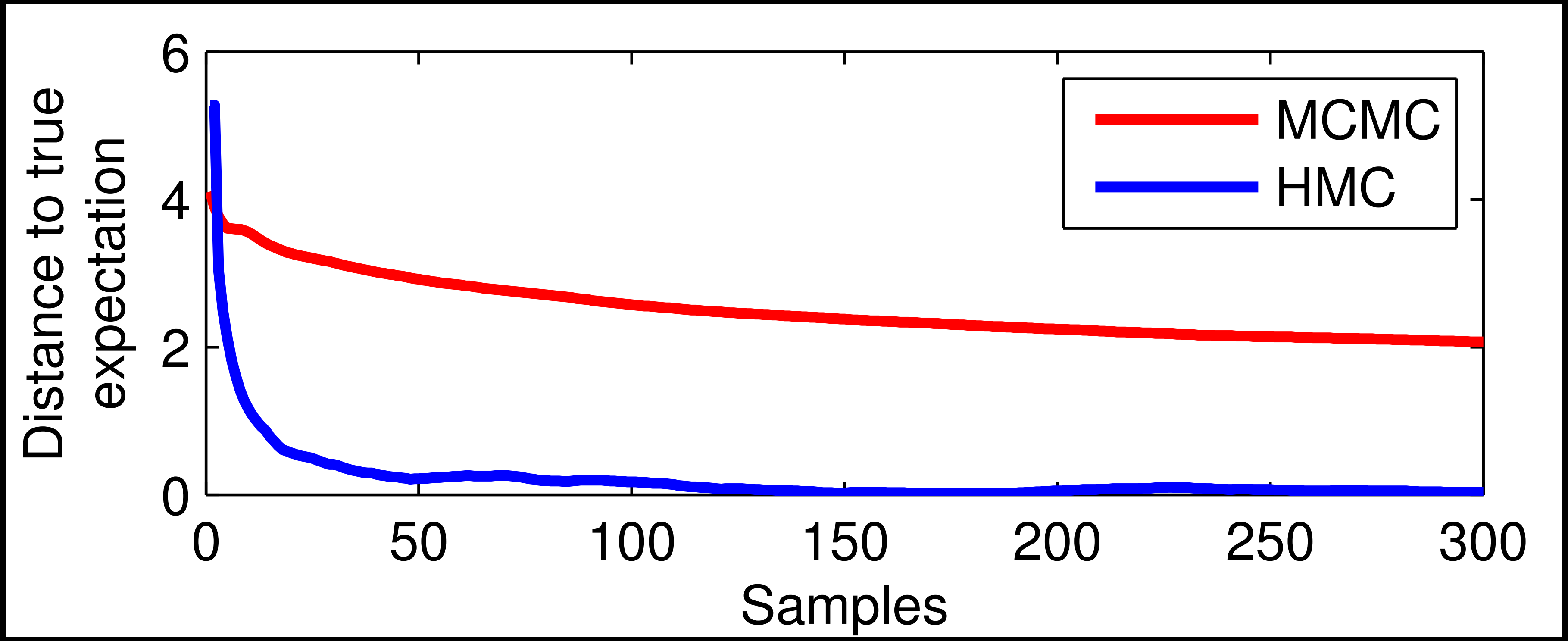
  - Numeric integration introduces conservation errors, so add an MH accept step.
  - Need the gradient of the score....

Automatic differentiation

- Since programs are compositions of functions, can compute derivatives automatically (remember the chain rule?)
  - A "non-standard interpretation" -- implemented by operator overloading.
  - Exact (to machine precision) gradient of score at each point. (No silly errors!)
  - Low overhead (roughly 2x in Bher).
  - Generalizes back-prop.

HMC via AD

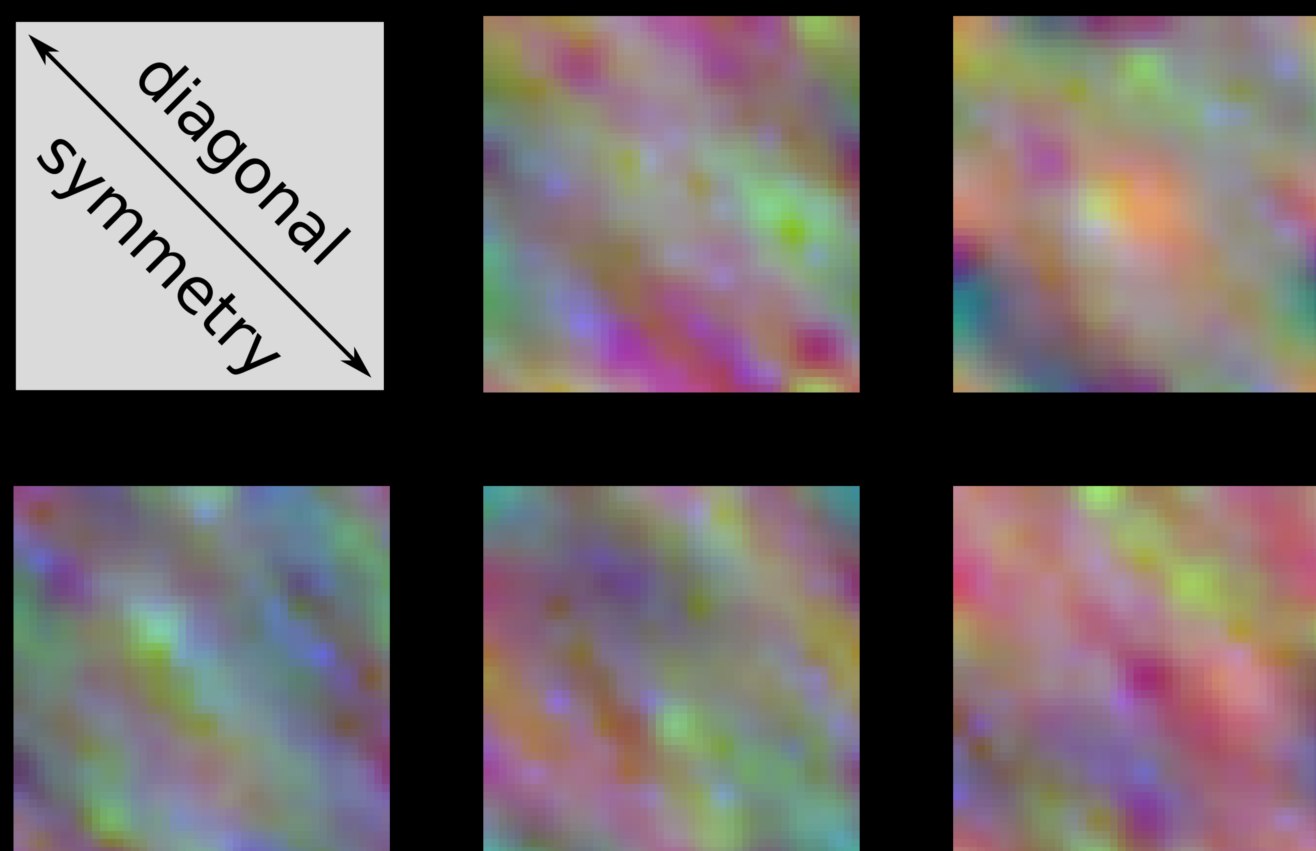
- We use AD to provide gradients needed in HMC.
  - HMC can be much more efficient.
  - Sampling an eccentric 3-dim gaussian:



Wingate, Stuhlmueller, Siskind, Goodman (under review)

HMC via AD

- We use AD to provide gradients needed in HMC.
  - Conditioning Perlin noise on symmetry:



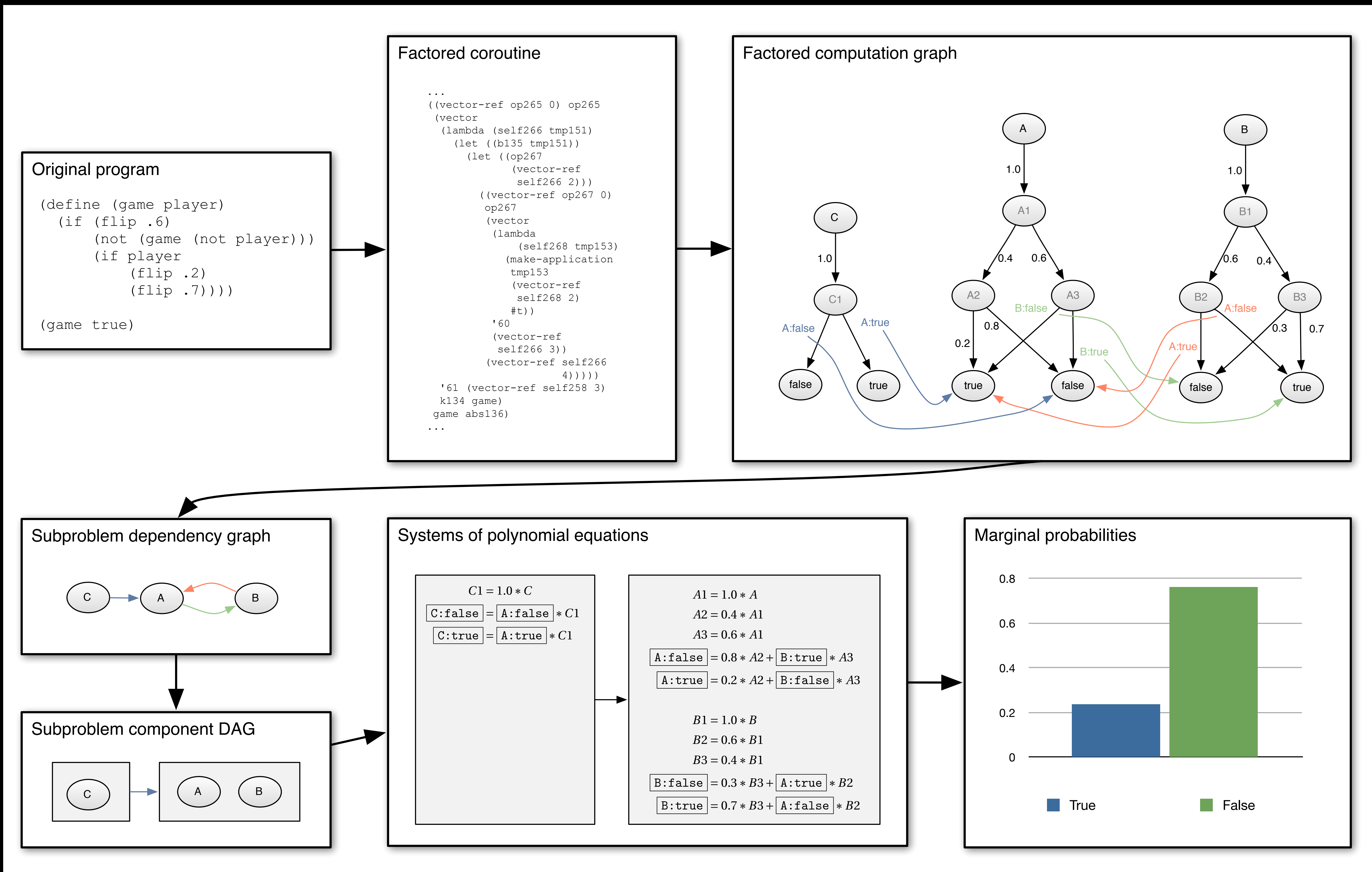
~1000 dims

Wingate, Stuhlmueller, Siskind, Goodman (under review)

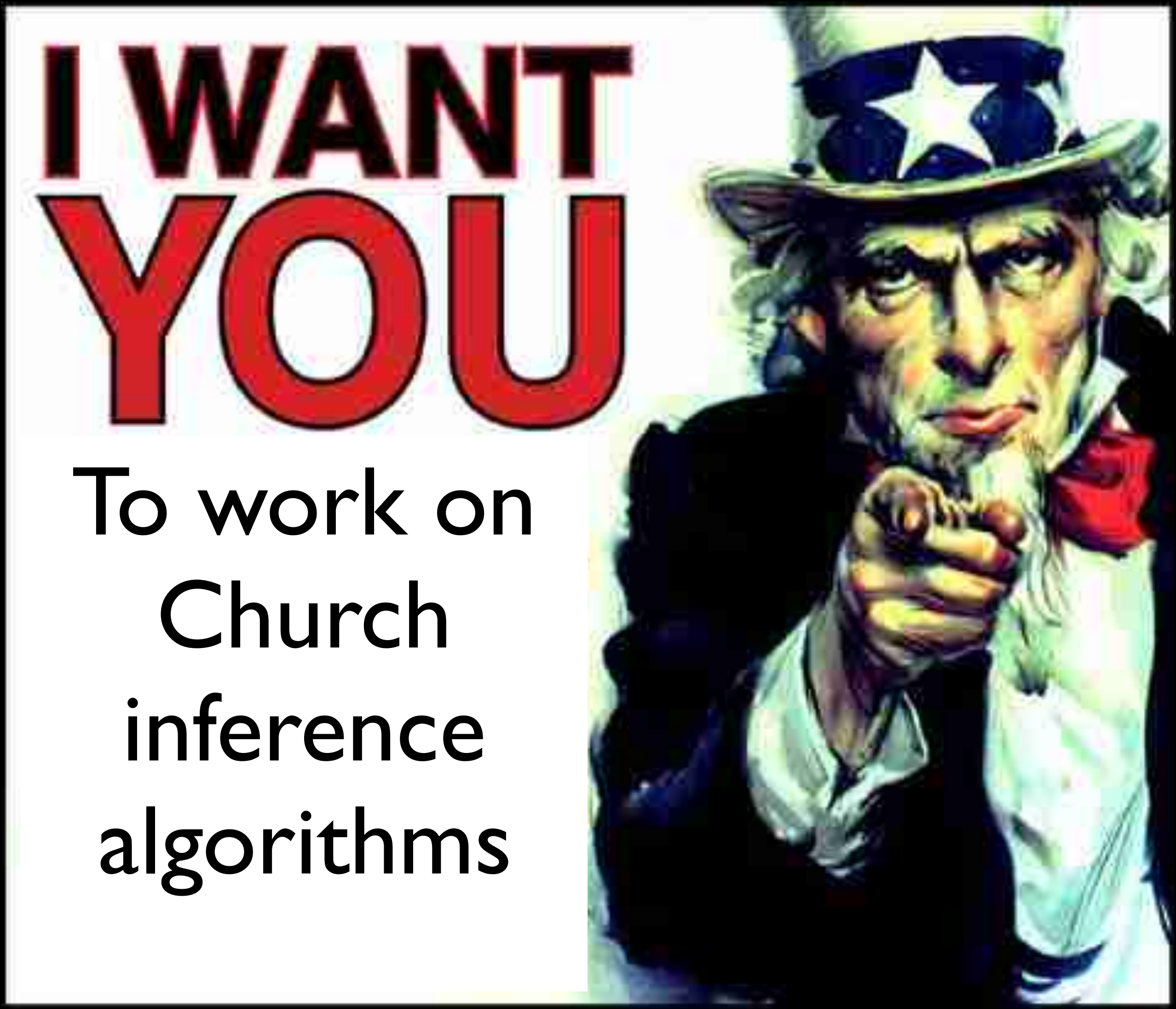
Other techniques

- Sequential Monte Carlo
    - Particle filter with rejuvenation,
    - Any model, and any sequentialization (specified via free variable in query).
- Dynamic programming (UDP/cosh)
    - Inference as marginalization,
    - Builds compact system of polynomial equations...

Other techniques



Many directions open..



Bonus: counterfactuals

- Structured models do more than specify distributions. See Pearl, next.
  - Can build a counterfactual operator in Church from the same “update” operator needed for MH.

```
(counterfactual
  ((dad-eyes 'brown))

  (define dad-eyes (uniform-draw '(blue brown)))
  (define mom-eyes (uniform-draw '(blue brown)))
  (if (or (eq? dad-eyes 'brown)
          (eq? mom-eyes 'brown)
          (flip 0.1))
      'brown
      'blue))
```

```
=>
(blue . brown)
```

(actual . counterf)

With Tobi Gerstenberg and Andreas Stuhlmueller.

Conclusion

- The PLoT: mental representations as stochastic functions.
- Church: an expressive probabilistic programming language suited for cognitive modeling.
- Next lectures: applications to social cognition, natural language, concept learning.