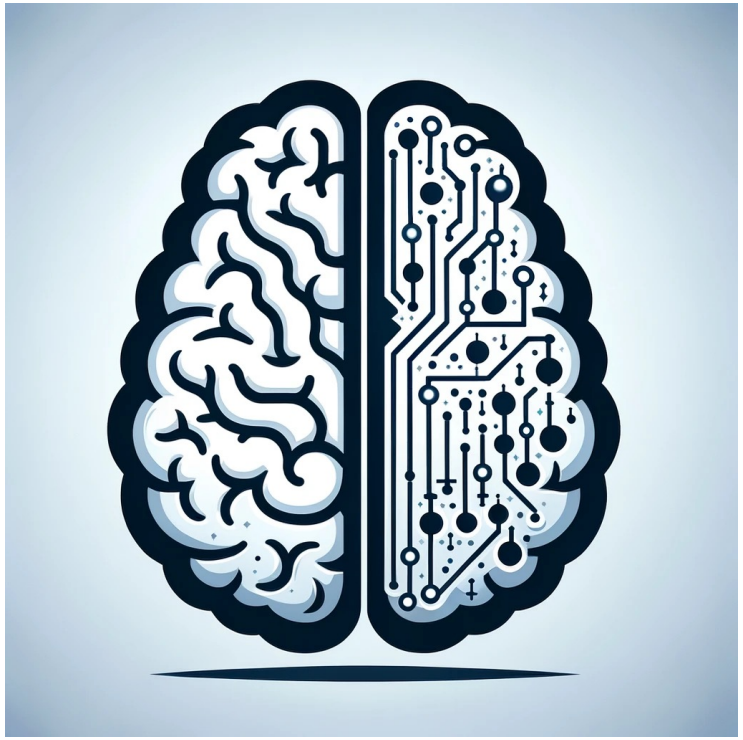
EN 601.473/601.673: Cognitive Artificial Intelligence (CogAI)



Lecture 7:
Intro to PPLs (WebPPL, Gen)

Tianmin Shu

Logistics

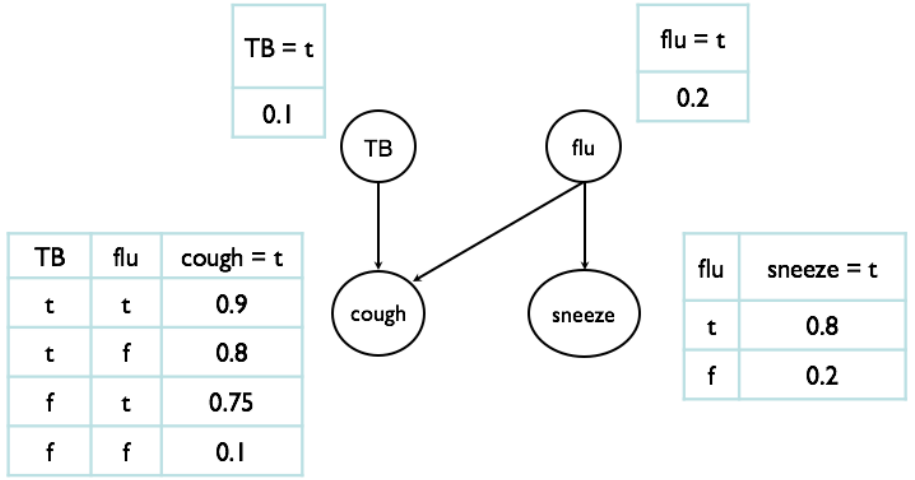
- Pset 1: due by Sunday, Feb 18, end of the day

Readings

- Probmods.org, chapters 2, 3, & 5
- From word models to world models (Sec. 2)
- The original Gen paper
- Three papers on real-world applications using Gen: data modeling and 3D computer vision (recommended background: computer vision)

Towards a probabilistic language of thought

- Bayesian networks
  - Mathematically convenient
  - Causal models
  - Hard to write code for inference algorithms



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

- Probabilistic programs
  - Have rich compositionality
    - Recall: composing programs for priors for the number game
  - Provide a complete specification of the domain (nothing hidden behind arrows)
  - Probabilistic programming languages (PPLs) provide tools for easily and flexibly implement efficient inference algorithms
  - PPLs for Bayesian models, DL frameworks (e.g., pytorch) for neural networks
  - Connected to natural language via LMs

```
var towModel = function() {
  var strength = mem(function (person) {return gaussian(50, 10)})

  var lazy = function(person) {return flip(0.1) }

  var pulling = function(person) {
    return lazy(person) ? strength(person) / 2 : strength(person) }

  var totalPulling = function (team) {return sum(map(pulling, team))}

  var winner = function (team1, team2) {
    totalPulling(team1) > totalPulling(team2) ? team1 : team2 }

  var beat = function(team1,team2){winner(team1,team2) == team1}

  condition(beat(['bob'], ['tom']))

  return strength('bob')
}
```

Universal probabilistic programming

WebPPL (Church 2.0)

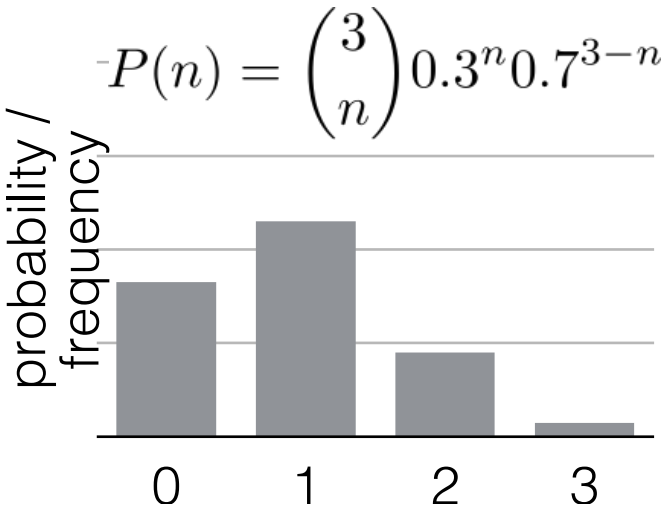
Named after Alonzo Church who is known for lambda calculus

Relationship between sampling and probability distributions

Random primitives:

```
var a = flip(0.3)
var b = flip(0.3)
var c = flip(0.3)
return a + b + c
```

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

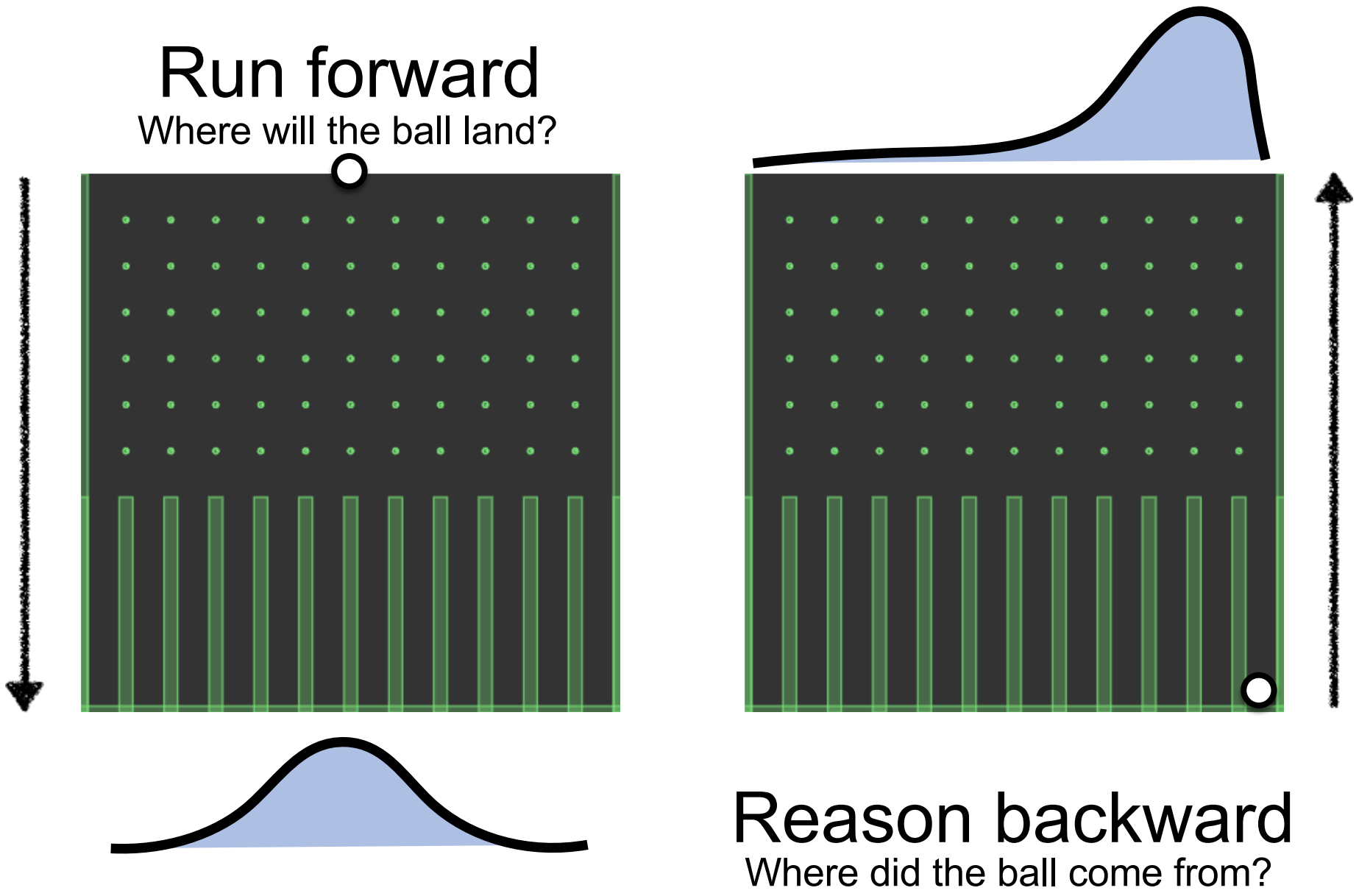


Sampling ≅ Distributions

Theorem: any computable distribution can be represented as the distribution induced by sampling from a probabilistic program

Goodman, Mansinghka, Roy, Bonawitz, & Tenenbaum (2008) Church: A language for generative models. Uncertainty in Artificial Intelligence

Inference in PPLs

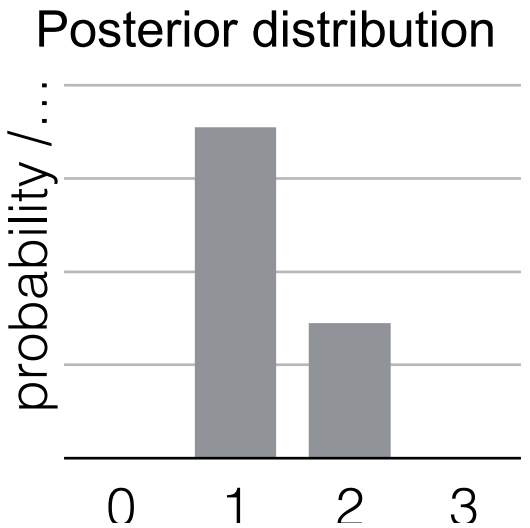


Inference in PPLs

Conditional inference:

```
Infer(
 function(){
  var a = flip(0.3)
  var b = flip(0.3)
  var c = flip(0.3)
    condition(a + b == 1)
  return a + b + c})
```

=> 1 0 0 1
=> 0 0 0 0
=> 1 0 1 0
=> T F F T
=> 2 0 1 1



- Rejection sampling is essentially a procedural definition of inference.
- Allowing inference with any computable condition statement <-> (almost) supporting any computable conditional distribution.

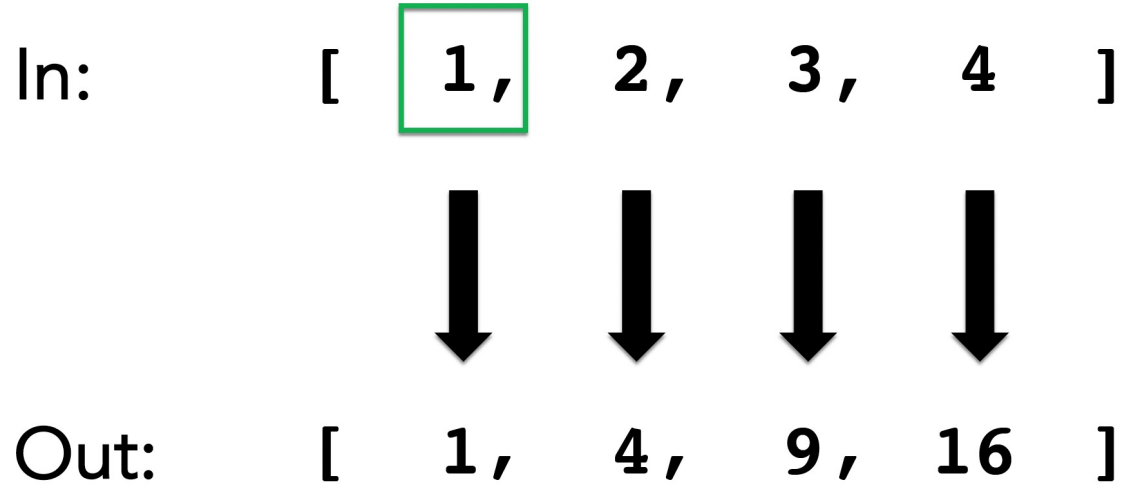
WebPPL basics

- Declare variables (random variables in a Bayesian model)
- Data formats: numbers, strings, logicals, objects, arrays
- if-then-statements
- defining functions
- Higher order functions
  - `map`

WebPPL basics

- The `map` function

```
var mySquare = function(x){return x*x}
var someNumbers = [1,2,3,4]
display(map(mySquare, someNumbers))
```



WebPPL basics

- WebPPL = purely functional programming language
- Can't write `for` loops or `while` statements
- Can create higher-order functions and recursive functions
  - For example: `map` = apply a function to each statement of a list (like a for loop)

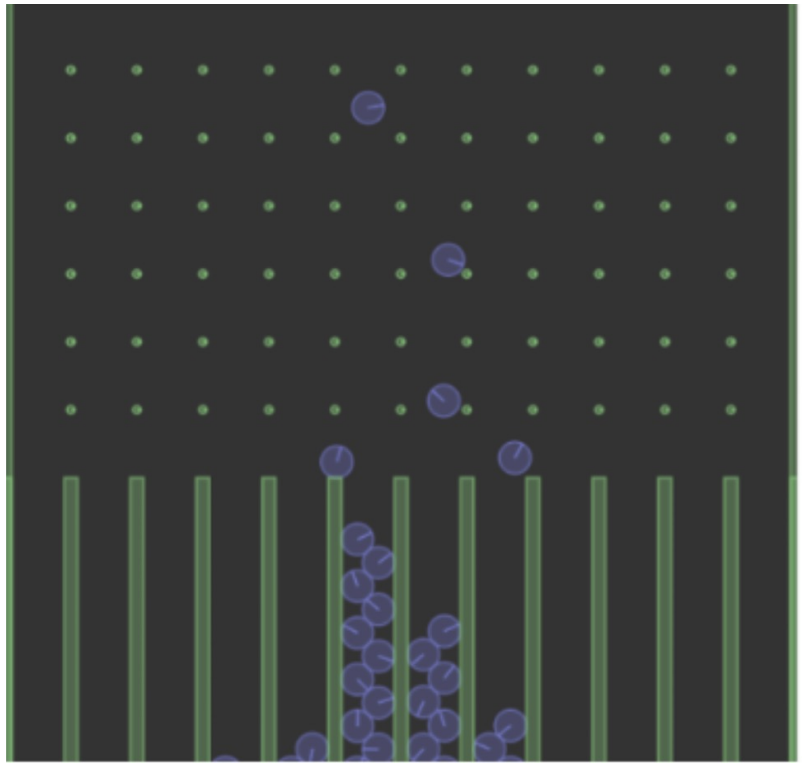
Inference in practice

- WebPPL supports a range of inference procedures
  - forward, rejection, enumerate, MCMC, …
- Don’t worry about the details or mechanisms of inference. Very nice, and very also limiting.
- But a good tool for parsimoniously describing rich generative model structures, and exploring the basic patterns of inference that emerge from the interaction of models and data.

- **Gen** (https://www.gen.dev/) supports inference programming: composing short (compact, understandable) programs for efficient inference, out of basic inference primitives.
- A much better tool for scalable engineering.

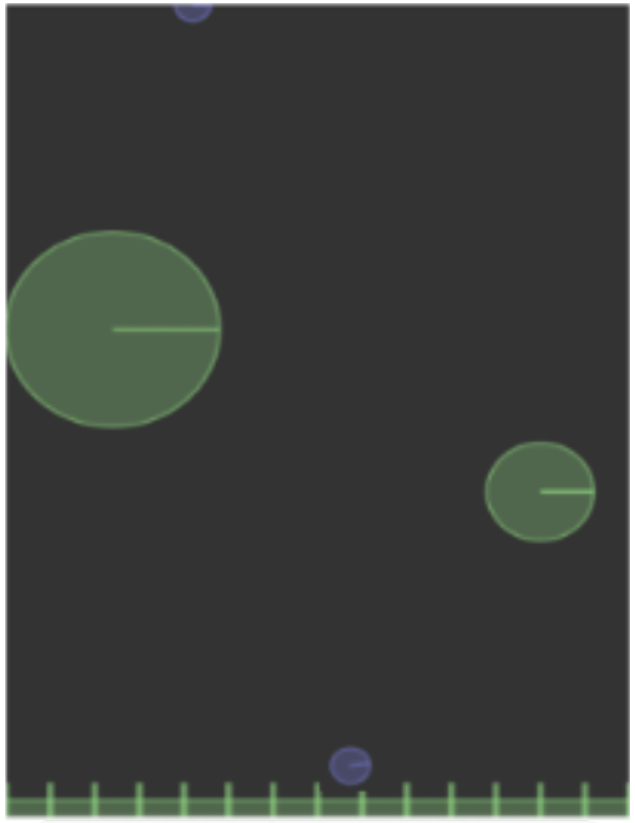
Run Forward

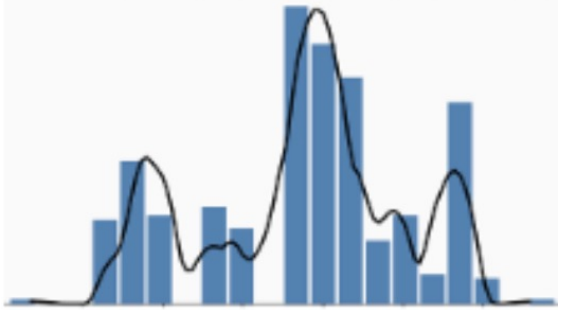
Generative model



Run Backward

Inference algorithm





WebPPL in Cognitive Science

(i) (ii) (iii)

Level N

Innate concepts

Entity

```
(define (make-entity property1 property2 … )
        (list property1 property2 … ))
```

Newtonian Dynamics

```
(define (run-dynamics entities forces
         init-cond steps dt)
  (if (= steps 0) '()
      (let* ((m (get-mass entities))
             (F (apply-forces forces entities))
             (a (/ F m))
             (new-cond (integrate init-cond
                                  a dt noise)))
        (pair new-cond (run-dynamics entities
                        forces new-cond (- 1 step) dt)))))
```

Level 2

Entity types

Puck: ○

```
(define puck (make-entity
                  pos shape mass vel … ))
```

Surface: □

```
(define surface (make-entity
                  pos shape friction … ))
```

Properties

Mass

```
(define (mass)
   (pair "mass" (uniform '(1 3 9))))
```

Friction

```
(define (friction)
   (pair "friction" '(uniform '(0 5 20))))
```

Force classes

Pairwise: ① ↔ ②

```
(define (pairwise-force c1 c2)
 (let* ((a (uniform-draw '(-1 0 1))))
 (lambda (o1 o2)
  (let ((r (euc-dist o1 o2)))
   (/ (* a del(o1,col(o1)) del(o2,col(o2)))
      (power r 2))))))
```

Global: ○→

```
(define (global-force)
  (let* ((d (uniform-draw compass-dir)))
   (lambda (o) (* k d))))
```

Level 1

Property values

- *: large mass*
- *: medium mass*
- *: small mass*
- *: high friction*
- *: no friction*

```
(define world-entities
 (map sample-values entity-list))
```

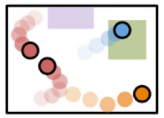
Force parameters

Force b/w reds: attract

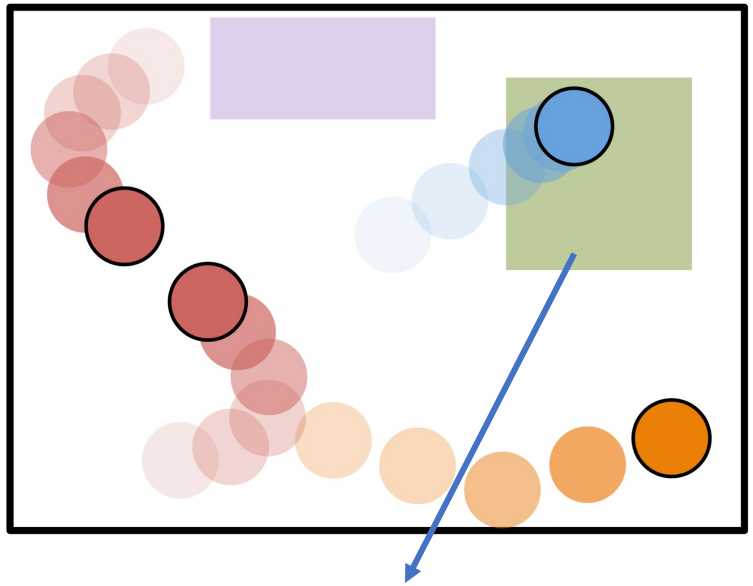
```
(define world-forces
 (map sample-parameters force-list)
```

Level 0 (data)

Initial conditions



```
(define scenario
 (let* (
       (init-cond (sample-init world-entities)))
   (run-dynamics world-entities
      world-forces init-cond steps dt)))
```



Friction?
Magnetic field?
Const. force (e.g.., wind)?

Ullman, Stuhlmüller, Goodman, & Tenenbaum (2018)
Learning physical parameters from dynamic scenes.
Cognitive Psychology

WebPPL in AI

Bayesian Inference of Temporal Task Specifications from Demonstrations

Ankit Shah
CSAIL, MIT
ajshah@mit.edu

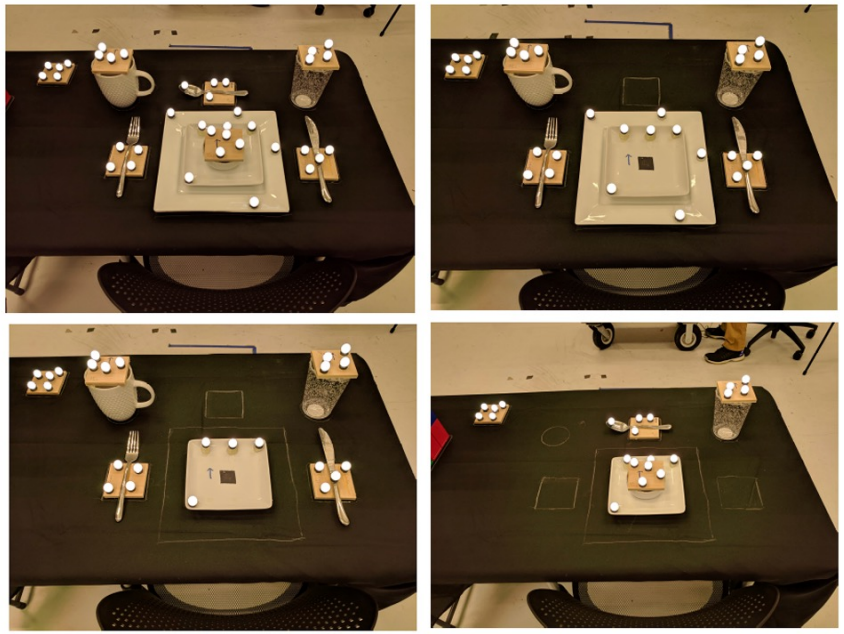
Pritish Kamath
CSAIL, MIT
pritish@mit.edu

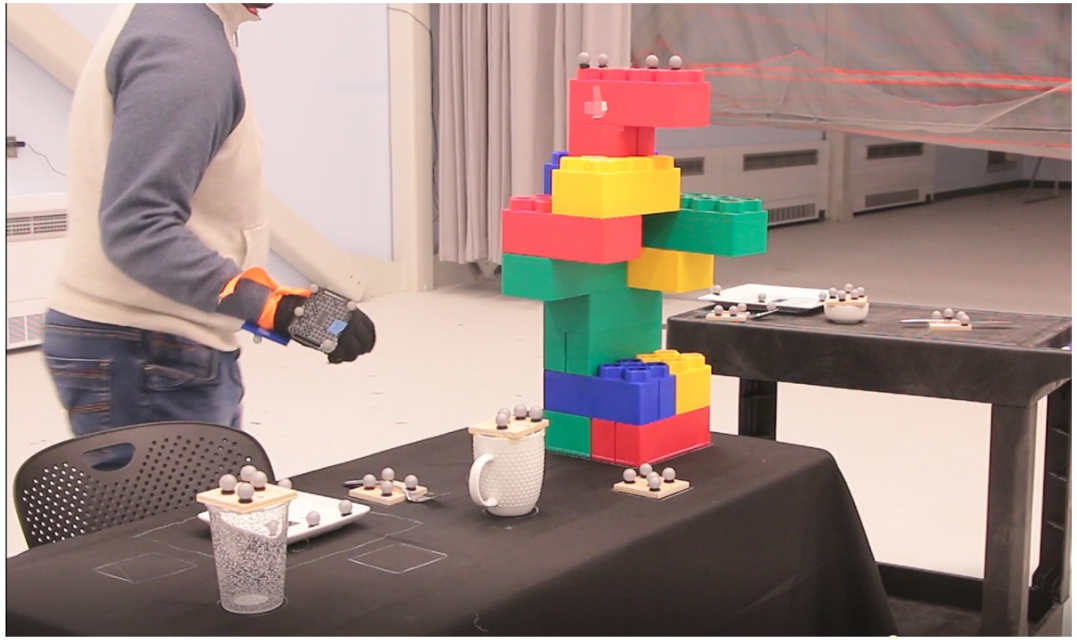
Shen Li
CSAIL, MIT
shenli@mit.edu

Julie Shah
CSAIL, MIT
julie_a_shah@mit.edu

NeurIPS 2018

WebPPL in AI





3.2 Specification Learning as Bayesian Inference

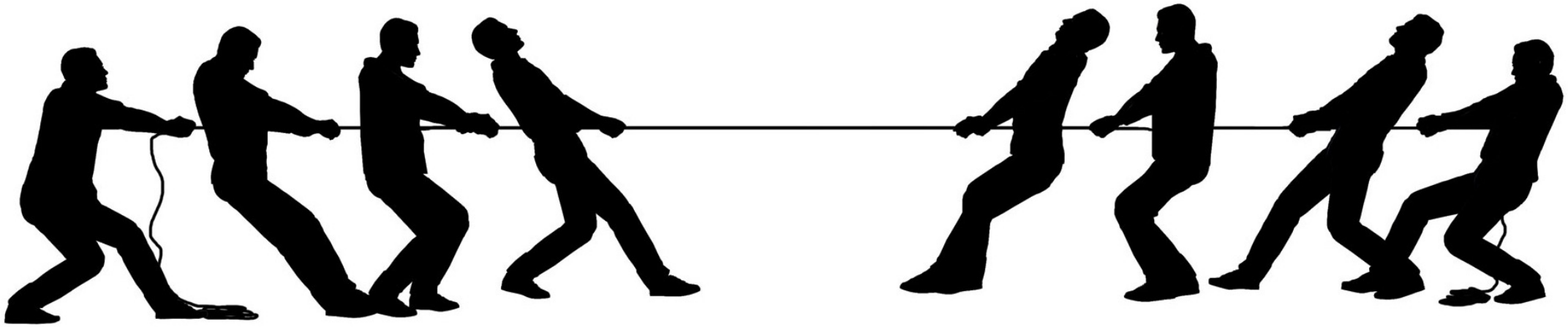
The Bayes theorem is fundamental to the problem of inference, and is stated as follows:

$$P(h \mid \mathbf{D}) = \frac{P(h)P(\mathbf{D} \mid h)}{\sum_{h \in \mathbf{H}} P(h)P(\mathbf{D} \mid h)} \tag{6}$$

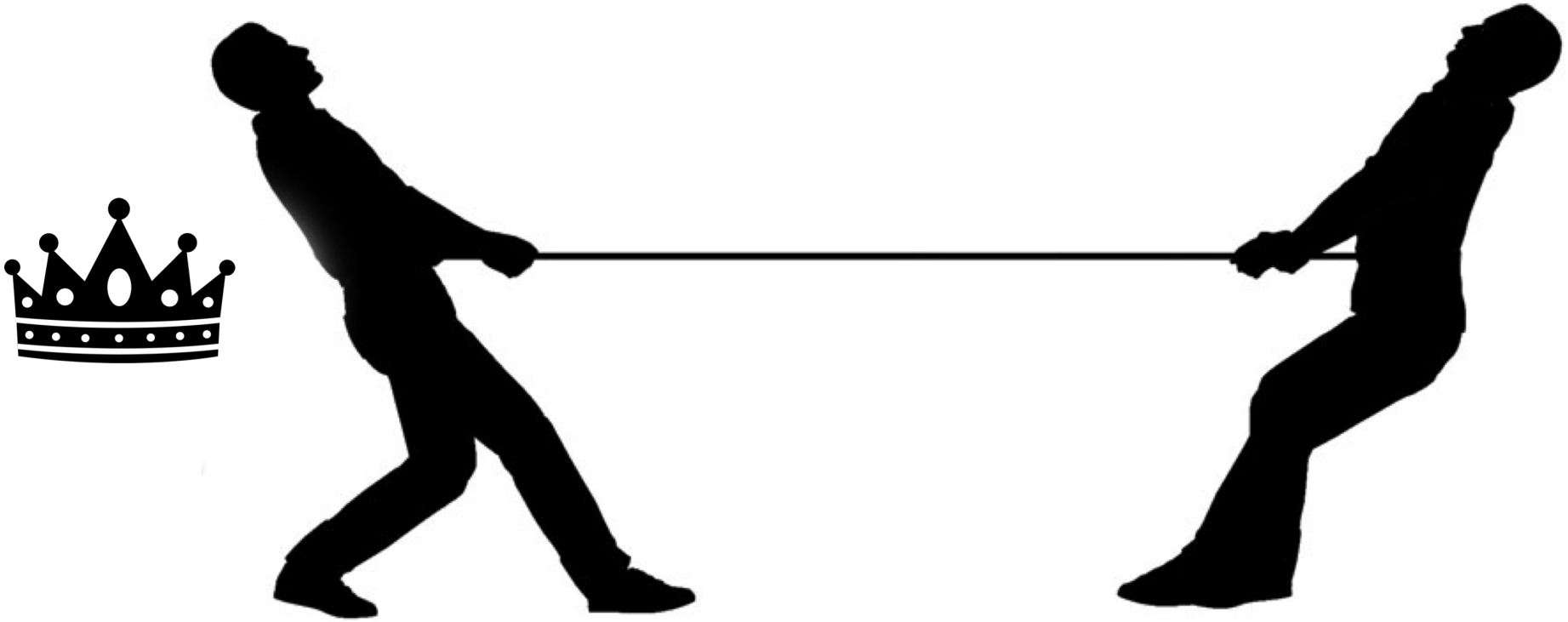
$P(h)$ is the prior distribution over the hypothesis space, and $P(\mathbf{D} \mid h)$ is the likelihood of observing the data given a hypothesis. Our hypothesis space is defined by $\boldsymbol{H} = \boldsymbol{\varphi}$, where $\boldsymbol{\varphi}$ is the set of all formulas that can be generated by the production rule defined by the template in Equation 2. The observed data comprises the set of demonstrations provided to the system by expert demonstrators (note that we assume all these demonstrations are acceptable). $\boldsymbol{D}$ represents a set of sequences of the propositions, defined by $\boldsymbol{D} = \{[\boldsymbol{\alpha}]\}$.

Project idea:
Concept learning in different domains using PPLs (Gen)

Probabilistic Tug of War



Inferring strengths based on the outcome



Tom?

Tom

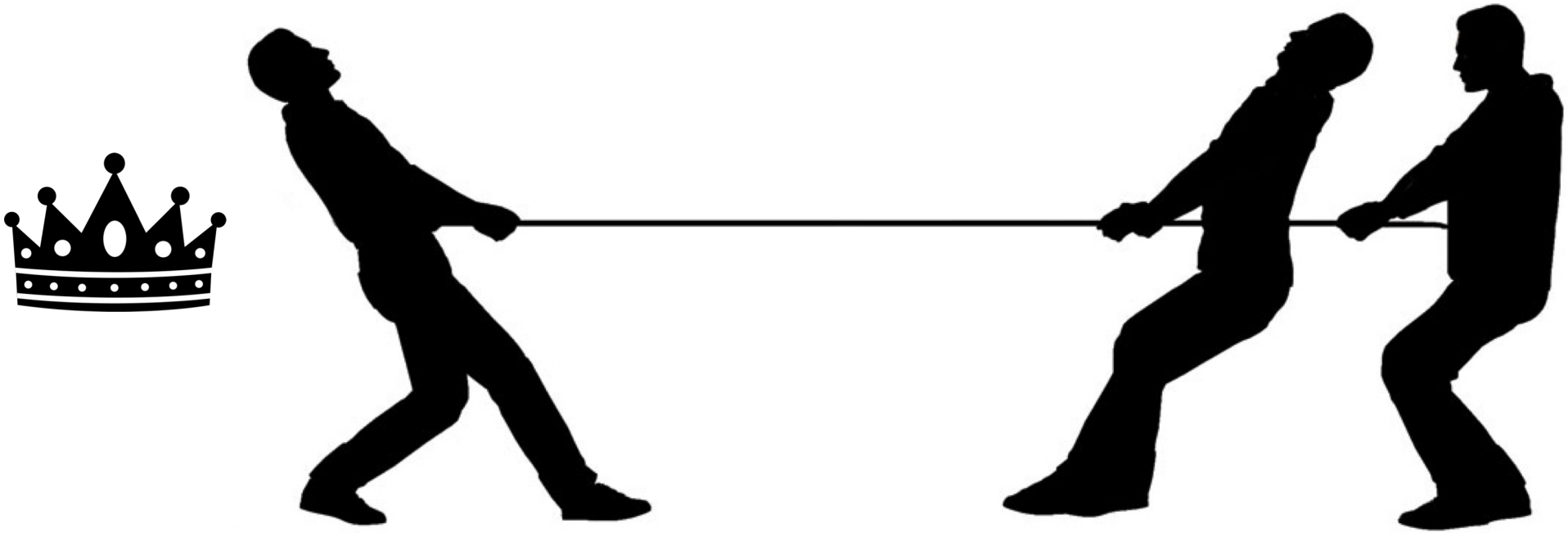
Steve

1 — very weak

50 — average

100 — very strong

Inferring strengths based on the outcome

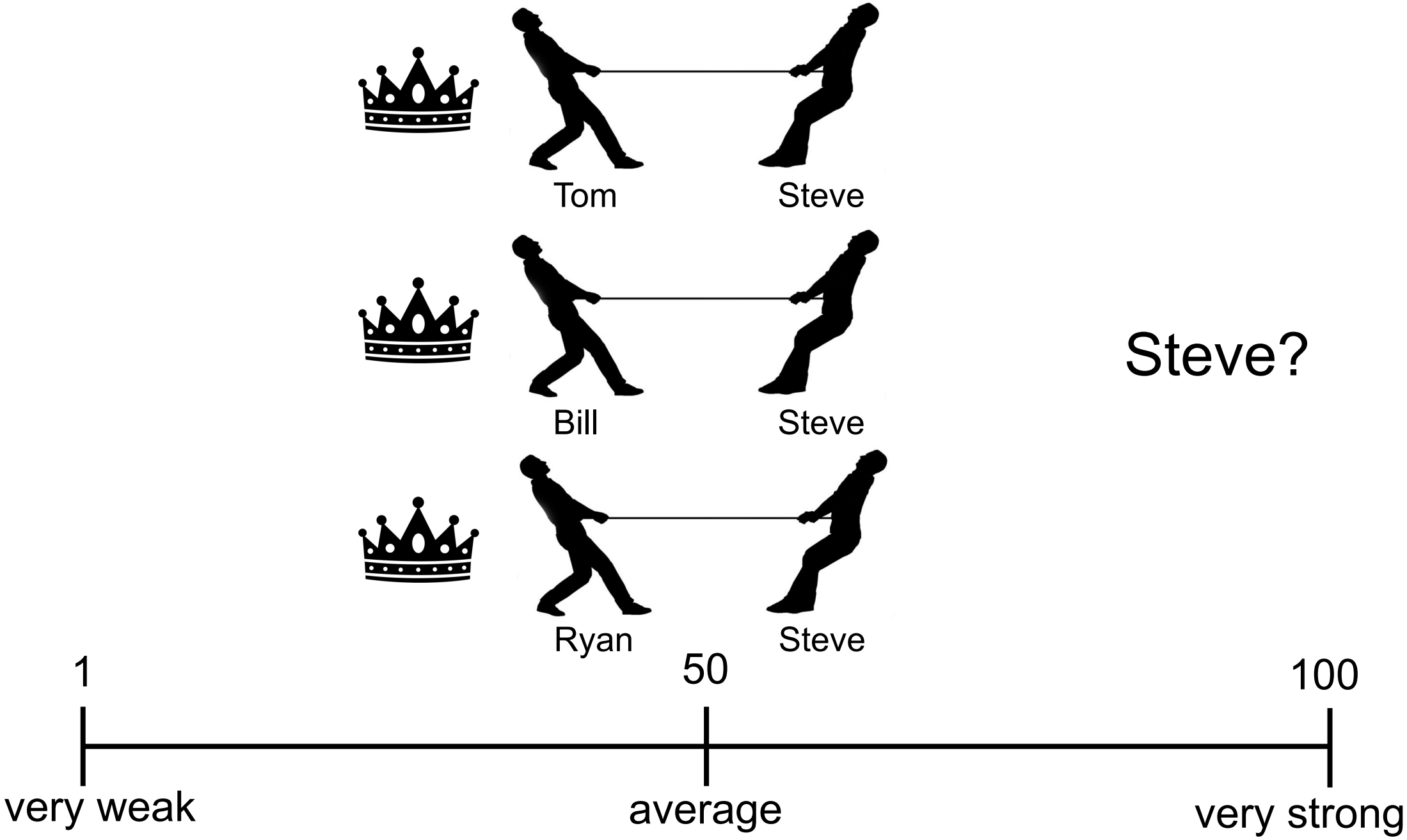


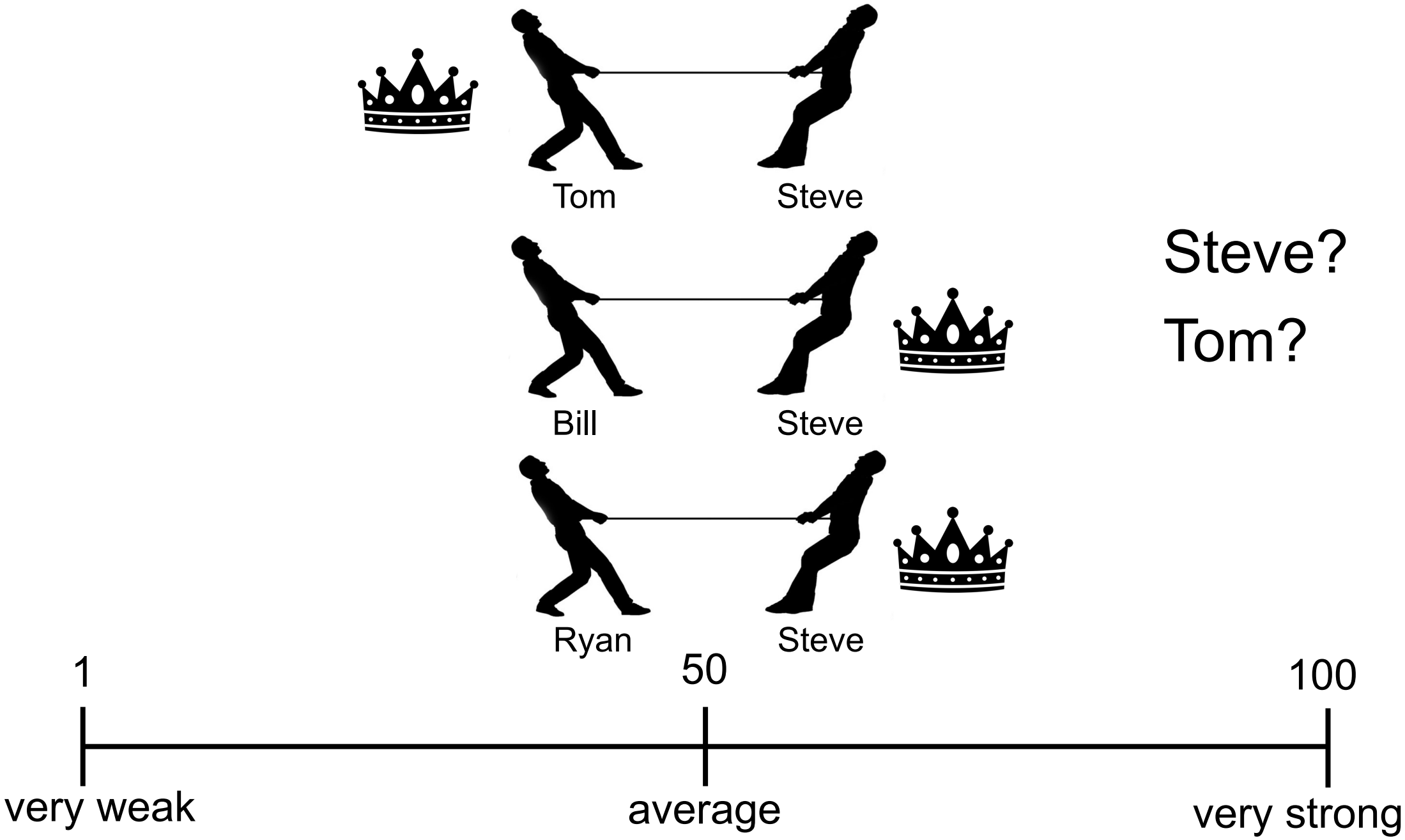
Tom?

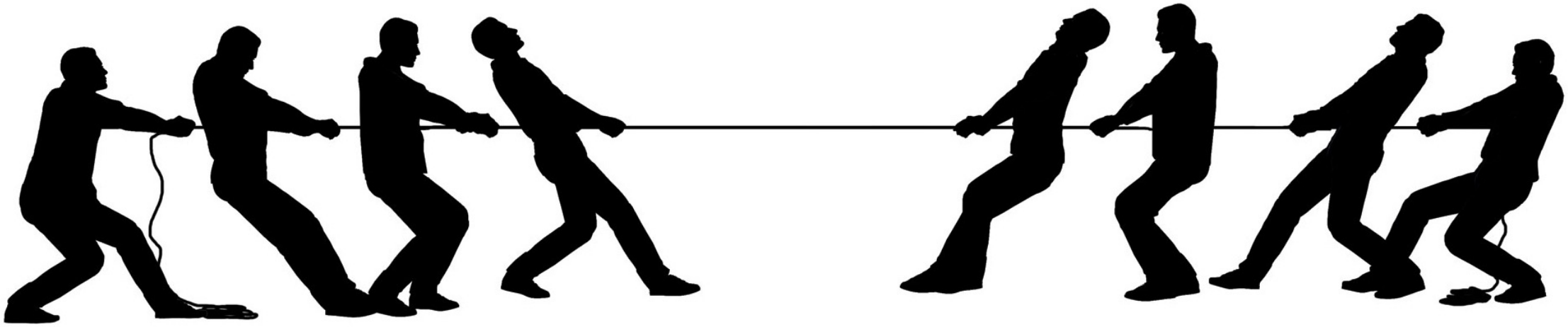
Tom Steve Nick

1 50 100

very weak average very strong


Tom
Steve
Bill
Steve
Ryan
Steve
Steve?
1
50
100
very weak
average
very strong


Tom
Steve
Bill
Steve
Ryan
Steve
Steve?
Tom?
1
50
100
very weak
average
very strong



A Bayesian network?

Concepts:

teams

person

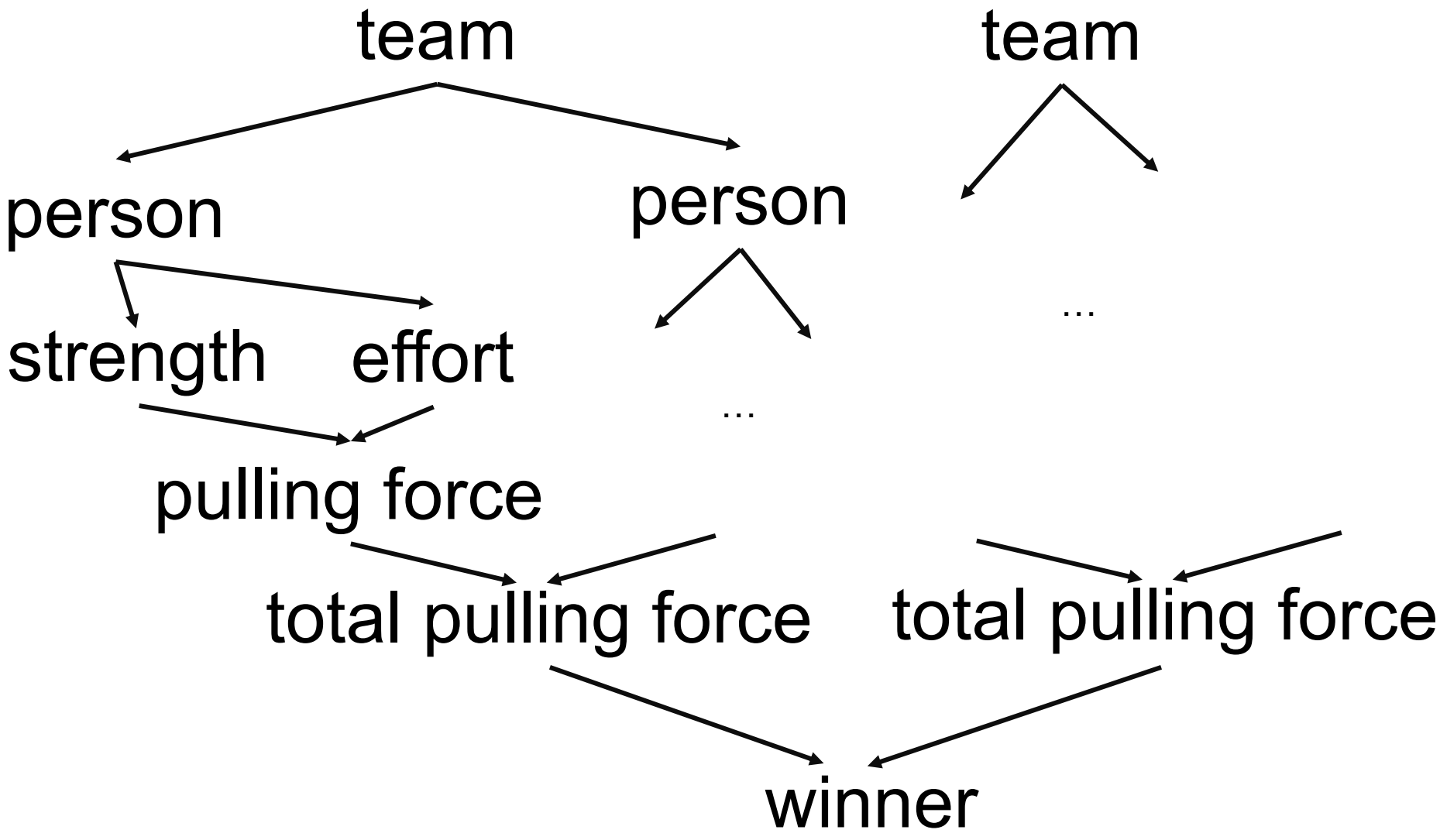
winner

strength

pulling force

effort

A Bayesian network?



A probabilistic program of tug of war

```
var towModel = function() {
  var strength = mem(function (person) {return gaussian(50, 10)})

  var lazy = function(person) {return flip(0.1) }

  var pulling = function(person) {
    return lazy(person) ? strength(person) / 2 : strength(person) }

  var totalPulling = function (team) {return sum(map(pulling, team))}

  var winner = function (team1, team2) {
    totalPulling(team1) > totalPulling(team2) ? team1 : team2 }

  var beat = function(team1,team2){winner(team1,team2) == team1}

  condition(beat(['bob'], ['tom']))

  return strength('bob')
}
```

https://probmods.org/chapters/conditioning.html

See WebPPL demo

Probabilistic language of thought hypothesis

- *The language of thought hypothesis* (LOTH) proposes that thinking occurs in a mental language.
- Probabilistic language of thought hypothesis:
- “Concepts have a language-like compositionality and encode probabilistic knowledge. These features allow them to be extended productively to new situations and support flexible reasoning and learning by probabilistic inference.”

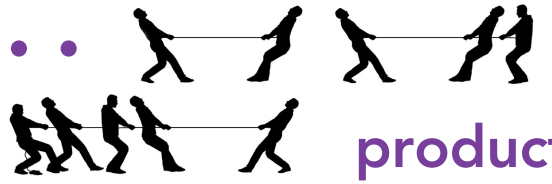
Fodor (1975). The language of thought

Goodman, Tenenbaum, & Gerstenberg (2015) Concepts in a probabilistic language of thought. *The Conceptual Mind: New Directions in the Study of Concepts*

“Concepts have a language-like compositionality and encode probabilistic knowledge. These features allow them to be extended productively to new situations and support flexible reasoning and learning by probabilistic inference.“

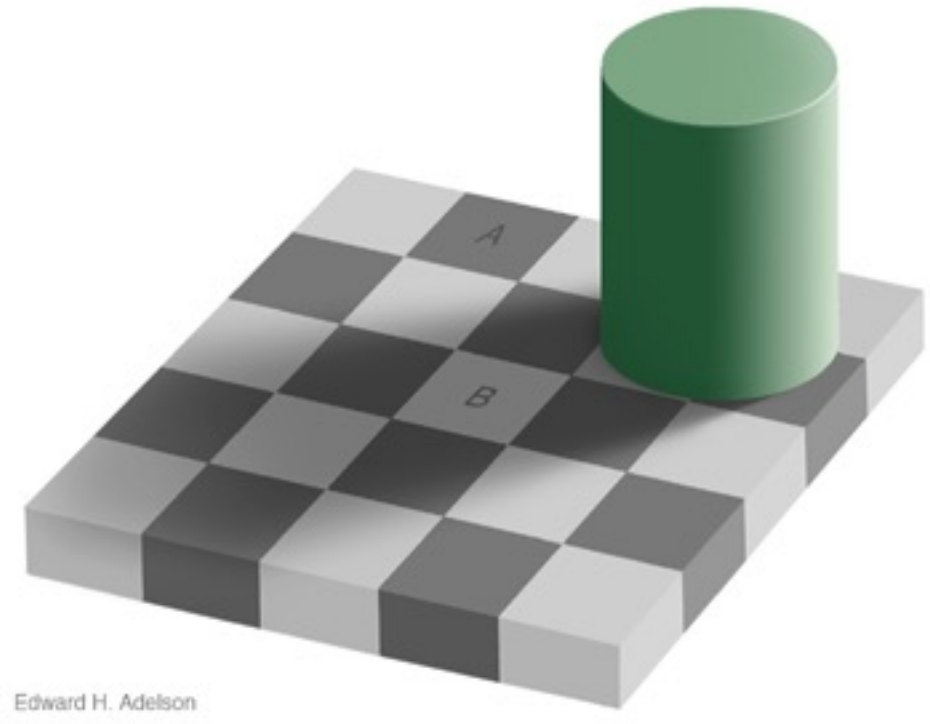
```
var towModel = function() {
  var strength = mem(function (person) {return gaussian(50, 10)})

  var lazy = function(person) {return flip(0.1) }

  var pulling = function(person) {
    return lazy(person) ? strength(person) / 2 : strength(person) }

  var totalPulling = function (team) {return sum(map(pulling, team))}

  var winner = function (team1, team2) {
    totalPulling(team1) > totalPulling(team2) ? team1 : team2 }

  var beat = function(team1,team2){winner(team1,team2) == team1}

  condition(beat(['bob'], ['tom']))

  return strength('bob')
}
```

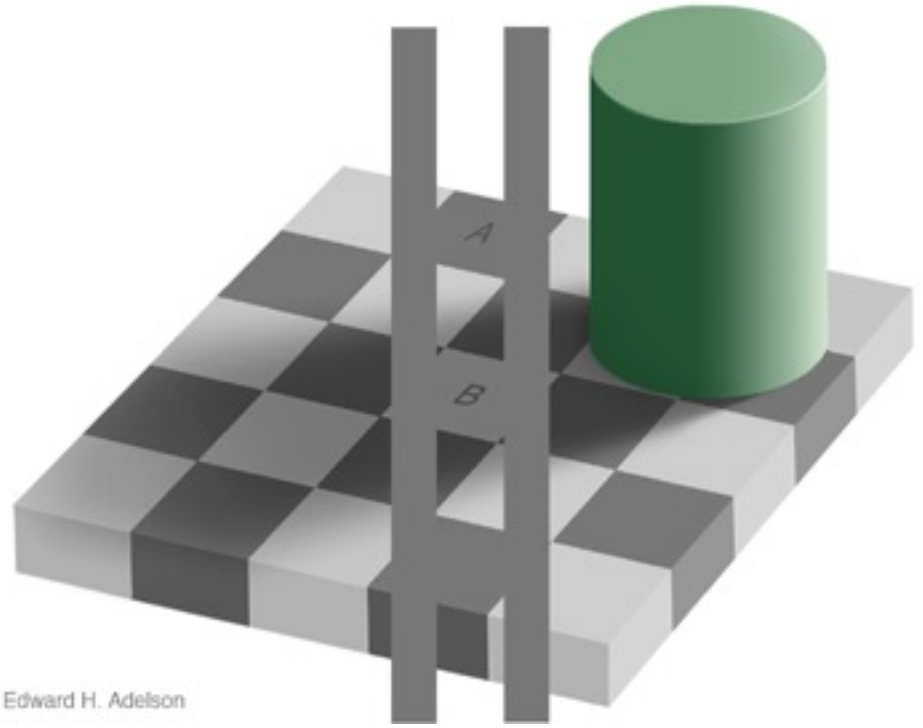
compositionality

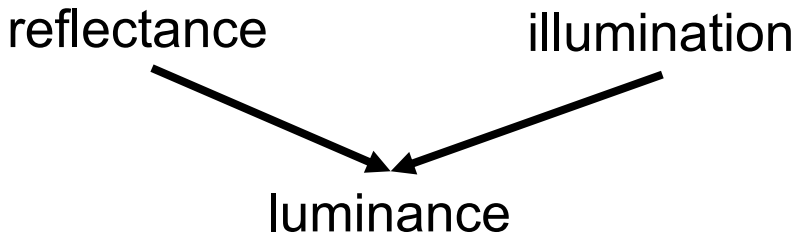


productivity

Explaining away using WebPPL







See WebPPL demo

https://probmods.org/chapters/conditional-dependence.html

More examples on probmods.org

Probabilistic Models of Cognition

by Noah D. Goodman, Joshua B. Tenenbaum & The ProbMods Contributors

This book explores the probabilistic approach to cognitive science, which models learning and reasoning as inference in complex probabilistic models. We examine how a broad range of empirical phenomena, including intuitive physics, concept learning, causal reasoning, social cognition, and language understanding, can be modeled using probabilistic programs (using the WebPPL language).

Contributors

This book is an open source project. We welcome content contributions (via GitHub)!

The ProbMods Contibutors are:
Noah D. Goodman (editor)
Joshua B. Tenenbaum
Daphna Buchsbaum
Joshua Hartshorne
Robert Hawkins
Timothy J. O'Donnell
Michael Henry Tessler

Citation

N. D. Goodman, J. B. Tenenbaum, and The ProbMods Contributors (2016). *Probabilistic Models of Cognition* (2nd ed.). Retrieved 2024-2-8 from `https://probmods.org/`
[bibtex]

Acknowledgments

We are grateful for crucial technical assitance from: Andreas Stuhlmüller, Tomer Ullman, John McCoy, Long Ouyang, Julius Cheng.

The construction and ongoing support of this tutorial are made possible by grants from the Office of Naval Research, the James S. McDonnell Foundation, the Stanford VPOL, and the Center for Brains, Minds, and Machines (funded by NSF STC award CCF-1231216).

Previous edition

The first edition of this book used the probabilistic programming language Church and can be found here.

Chapters

Reasoning with language

From Word Models to World Models:
Translating from Natural Language to the Probabilistic Language of Thought

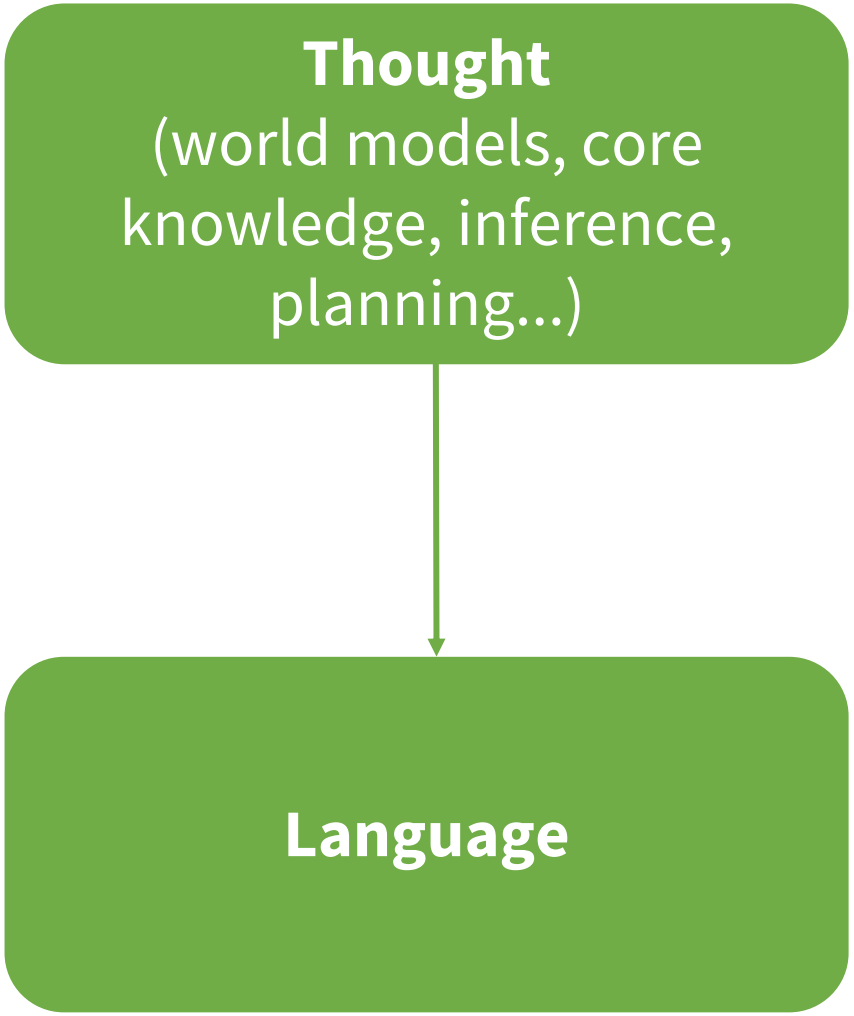
Lionel Wong[1⋆], Gabriel Grand[1⋆], Alexander K. Lew[1], Noah D. Goodman[2], Vikash K. Mansinghka[1], Jacob Andreas[1], Joshua B. Tenenbaum[1]

⋆ *Equal contribution.*

[1]MIT, [2]Stanford

Reasoning via large language models

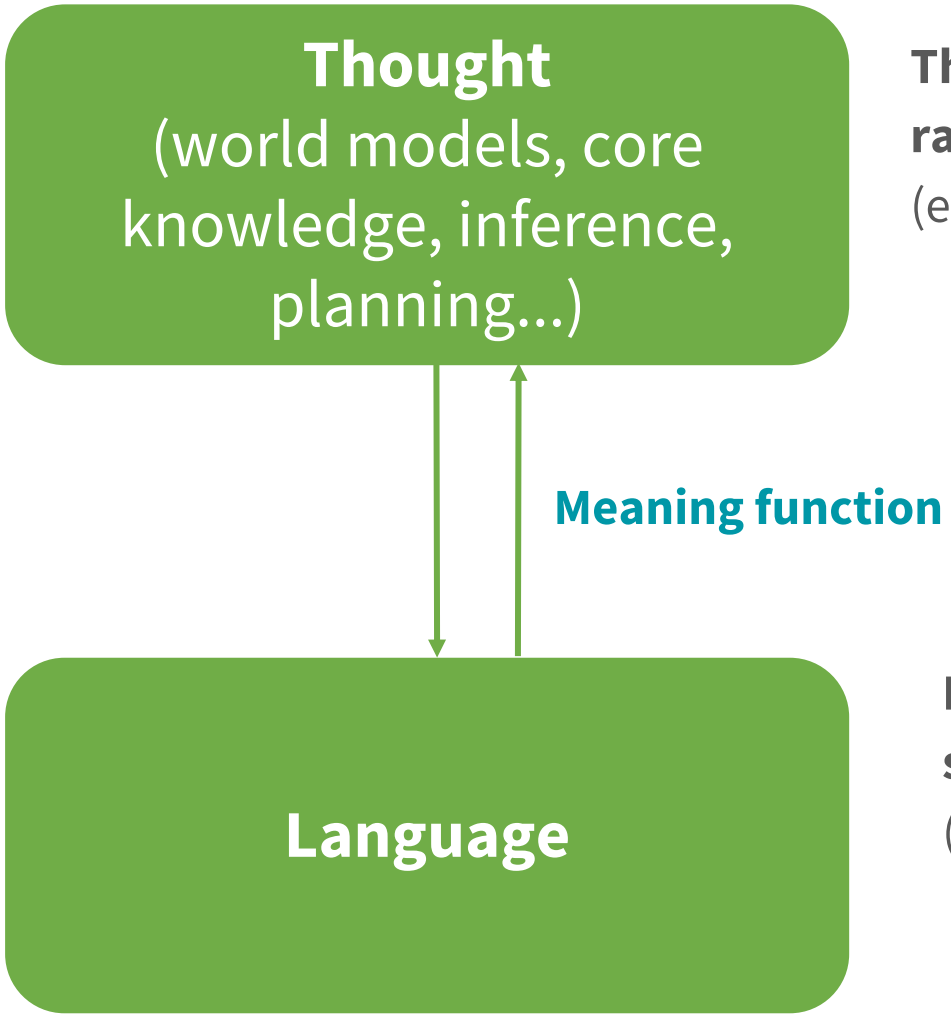
The "LLM scaling" hypothesis: invert the entire generative process to learn *thought* from the distribution of language



Distributional language modeling: learn linguistic structure from the statistical distribution of language
(recurrent neural networks, sequence modeling, latent semantic analysis, distributional clustering, topic models word2vec, glove, LSTMs, ...)

[Slides credits: Gabe Grand & Lio Wong]

Our proposal: translating from language into thought

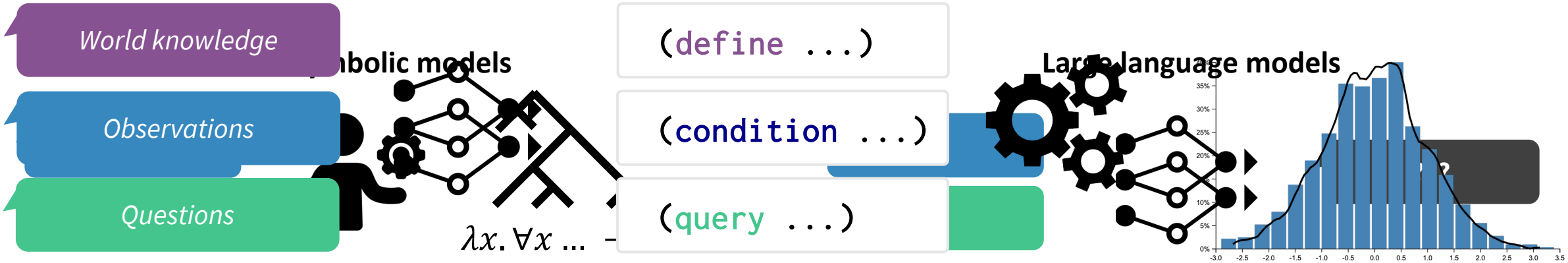


Thought requires systems for **goal-directed world modeling, rational inference, and decision making.**
(eg. Anderson, Miller & Johnson-Laird, Gentner, many others)

Language involves **meaning functions that map from external systems of signs** into **structured internal representations.**
(eg. Frege, 1893; Heim and Kratzer, 1998; Pietrovski, 2005)

[Slides credits: Gabe Grand & Lio Wong]

Approaches to language-informed thinking



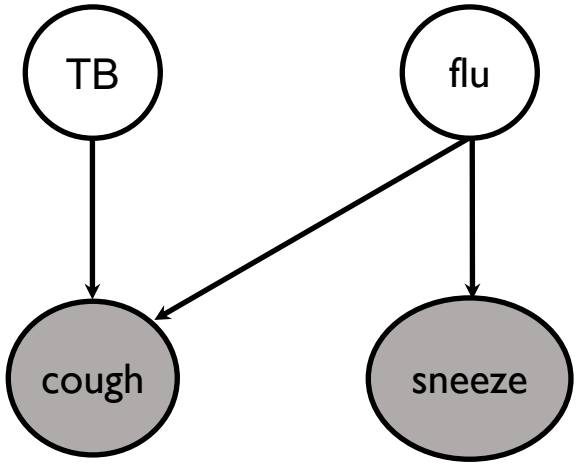
Natural language	Meaning function	Probabilistic language of thought	Inference function	Distributions over possible worlds

Neural — Symbolic

[Slides credits: Gabe Grand & Lio Wong]

Probabilistic inference with Bayesian networks

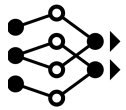
- Given:
  - Bayesian network that defines a joint distribution
    $$P(x_1, x_2, \ldots, x_n)$$
  - Condition / evidence / observation
    $O = o, O \subseteq X$ is a subset of (observed) variables
  - Query:
    $Q \subseteq X$ is a subset of (latent) variables
- Compute:



$P(Q = q | O = o)$ for all values q

Implementation

We created a minimal implementation of our framework using off-the-shelf tools.



Meaning function: OpenAI's **Codex** model, a large statistical language model trained jointly on language and code. Codex is *smaller* than GPT-4. We expect a range of code-trained LLMs to perform comparably in this role.

Inference function: We use **Church**, a probabilistic programming language that implements *simple,* universal sampling-based inference methods to perform inference across our models. Church has a minimalistic lambda calculus syntax that is evocative of formal notations from linguistics.

Our goal is to *demonstrate* how this framework might broadly interface between language and a range of core cognitive domains.

[Slides credits: Gabe Grand & Lio Wong]

(A) Generative world model

```
(define strength (mem (lambda (player)
  (gaussian 50 20))))

(define laziness (mem (lambda (player)
  (uniform 0 1))))

(define (team-strength team)
  (sum (map (lambda (player)
    (if (flip (laziness player))
        (/ (strength player) 2)
        (strength player)))
  team)))

(define (won-against team-1 team-2)
  (> (team-strength team-1) (team-strength team-2)))
```

[Slides credits: Gabe Grand & Lio Wong]

(A) Generative world model

```
(define strength (mem (lambda (player)
  (gaussian 50 20))))
```


```
(define laziness (mem (lambda (player)
  (uniform 0 1))))
```

. . .

(B) Translation examples for LLM prompting

John and Mary won against Tom and Sue.

```
(condition (won-against '(john mary) '(tom sue)))
```

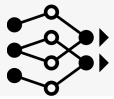
Sue is very strong!

```
(condition (> (strength 'sue) 75))
```

If Sue played against Tom, who would win?

```
(query (won-against '(sue) '(tom)))
```

[a new cond. statement / query]



. . .

[Slides credits: Gabe Grand & Lio Wong]

(C) Natural language

Josh won against Lio.

Josh proceeded to claim victory against Alex.

Even working as a team, Lio and Alex still could not beat Josh.

How strong is Josh?

What are the odds of Gabe beating Josh?

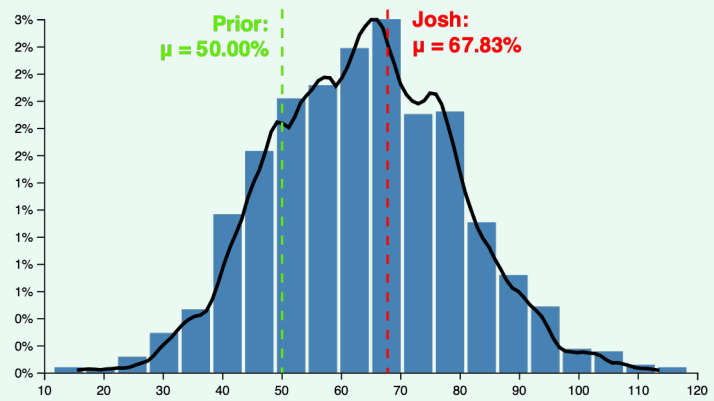
(D) Language of thought

```
(condition (won-against '(josh) '(lio)))
```

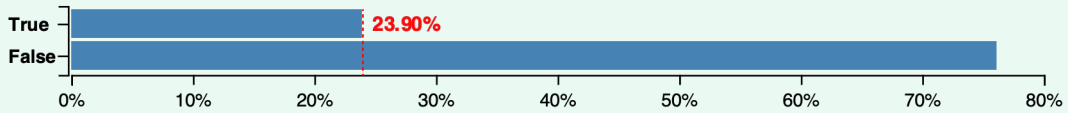
```
(condition (won-against '(josh) '(alex)))
```

```
(condition (not (won-against '(lio alex)
                             '(josh))))
```

```
(query (strength 'josh))
```




```
(query (won-against '(gabe) '(josh)))
```



[Slides credits: Gabe Grand & Lio Wong]

The faculty team is made up of four players: Jacob, Josh, Noah, and Vikash.

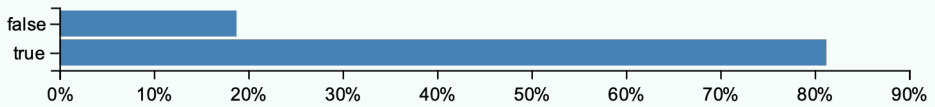
```
(define faculty-team '(jacob josh noah vikash))
```

The student team consists of Alex, Gabe, and Lio, plus their labmates, Ben and Ced.

```
(define student-team '(alex gabe lio ben ced))
```

Is Gabe stronger than the weakest player on the faculty team?

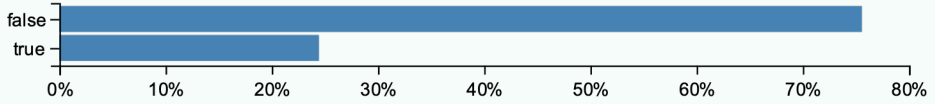
```
(query
  (stronger-than?
    'gabe (argmin strength faculty-team))
```



All of the faculty are pretty strong.

```
(condition
  (all
    (map
      (lambda (player)
        (> (strength player) 60))
     faculty-team)))
```

Is Gabe stronger than the weakest player on the faculty team?



Constructing a new world model

A. Prompt, containing unrelated example world model

```
;; We define a probabilistic model in Church of the
following scenario.
;; At any given time, about 1% of the population has lung
cancer,
;; 20% have a cold,  10% have a stomach flu, and 0.5% have
TB.
(define lung-cancer (mem (lambda (person) (flip 0.01))))
(define cold (mem (lambda (person) (flip 0.2))))
(define stomach-flu (mem (lambda (person) (flip 0.1))))
(define TB (mem (lambda (person) (flip 0.005))))

;; If you have a cold, there's a 50% chance you have a
cough.
;; 30% of people with lung cancer have a cough, and 70%
with TB.
;; There's also a small chance you have a cough even if
you're otherwise ;; healthy.
(define cough (mem (lambda (person)
  (or (and (cold person) (flip 0.5))
        (and (lung-cancer person) (flip 0.3))
        (and (TB person) (flip 0.7))
        (flip 0.01)))))

;; Whether a person coughs during a particular visit to the
doctor's
;; office depends on whether they have a cough, and a bit
of random
;; chance. Note that this will differ each time they go to
the doctor's
;; office, so we do not use `mem` (which memoizes the
result).
(define coughs-on-particular-visit
  (lambda (person) (and (cough person) (flip 0.7))))
```

B. Defining a new world model from scratch

```
;; Now, let's define a different probabilistic model of the following
scenario.
;; It is totally unrelated to the previous model and does not reference
the functions above.
```

First, strength levels vary widely from person to person.

```
(define strength (mem (lambda (person) (normal 100 20))))
```

Furthermore, each person has a percentage of the time that they are lazy.

```
(define laziness (mem (lambda (person) (uniform 0 1))))
```

The strength of a team is the combined strength of its members, except that in any given match, each player may decide to be lazy, and thus contribute only half of their strength.

```
(define team-strength
  (lambda (members)
    (apply + (map (lambda (member)
                    (if (flip (laziness member))
                        (/ (strength member) 2)
                        (strength member)))
                  members))))
```

Whether one team beats another just depends on which team pulls stronger that match.

```
(define team-beats-team
    (lambda (team1 team2)
       (> (team-strength team1) (team-strength team2)))))
```


[Slides credits: Gabe Grand & Lio Wong]

Perceptual reasoning

Generative model over static scenes

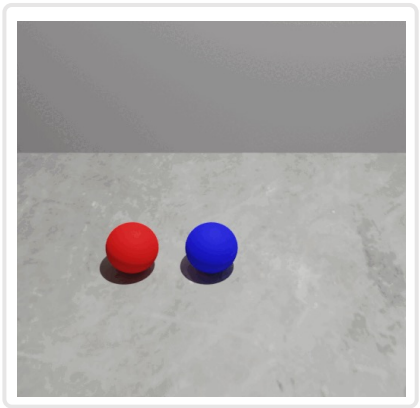
```
(define generate-object
  (mem (lambda (obj-id) (list
    (pair 'object-id obj-id)
    (choose-shape obj-id)
    (choose-color obj-id)))))

(define choose-num-objects ...) (define
generate-objects-in-scene ...)...
```

Graphics rendering engine

Sampled worlds

Vision as inverse graphics
(e.g., Kersten & Yuille, 1996;
Mansinghka et. al, 2013)

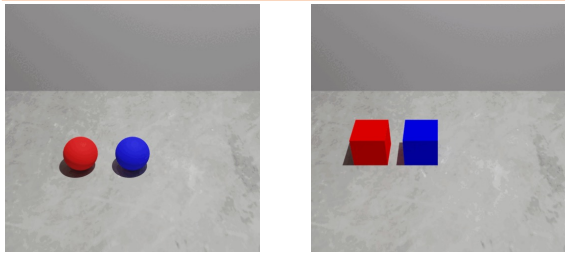
Physical reasoning



Generative model over dynamic scenes

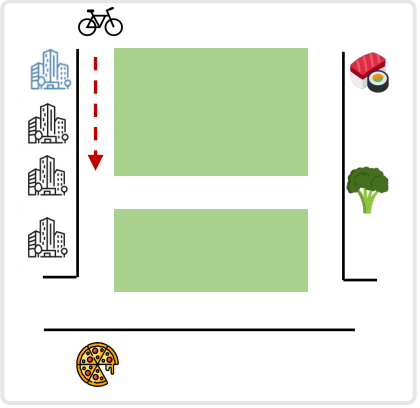
```
.(define generate-object
    (mem (lambda (obj-id)
    (list (pair 'object-id obj-id)
    (choose_shape obj-id)
    (choose_color obj-id)
    (choose_mass obj-id)...))))
(define generate-initial-scene-state...)
....
```

Physics engine



Intuitive physics as a
mental physics engine
(e.g., Battaglia et. al, 2013)

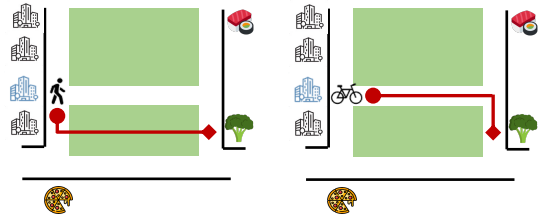
Social reasoning



Generative model over agents

```
(define restaurants (list 'sushi 'pizza
'vegetarian))

(define restaurant_utility (mem (lambda
(agent-id restaurant_type)
  (uniform-draw (list (gaussian
POSITIVE_UTILITY_MEAN UTILITY_VARIANCE)
  (gaussian NEGATIVE_UTILITY_MEAN
UTILITY_VARIANCE))))))
```

Planning engine



Intuitive social reasoning
as inverse planning
(e.g. ,Baker, Saxe, and Tenenbaum, 2009)

[Slides credits: Gabe Grand & Lio Wong]

Perceptual reasoning

Observations about the world

There is at least one red mug in this scene.

Condition statements

```
(condition
(>= (length
((filter-shape 'mug)
((filter-color red)
(objects-in-scene
 'this-scene)))) 1))
```

Questions about the world

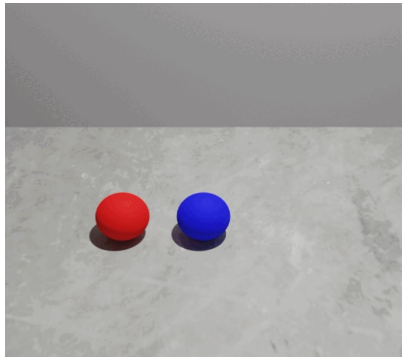
How many mugs are there?

Query statements

```
(query
  (length
    ((filter-shape 'mug)
     (objects-in-scene
      'this-scene))))
```

Inference over scenes using inverse graphics

Physical reasoning



Imagine that there is a red ball that is pretty heavy.

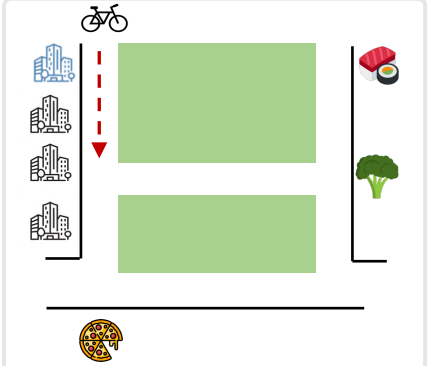
```
(condition ((get_singleton_object
(lambda (object) (and
((is-color? red) object)
(((is-shape? 'sphere) object)
(> (get-attribute object 'mass) 2.5)))))
```

How fast will the blue ball move if it's bumped by the red one?

```
(query …
   (get_attribute
event 'subject_final_velocity))
   (filter_events (lambda (e)
   (and (is_event? 'collision
    e...)
```

Inference over physics using physics engine

Social Reasoning



Alex loves sushi but hates pizza, and he brought his bike to work today.

```
(condition
  (and (loves? 'alex 'sushi)
       (hates? 'alex 'pizza)
       (has-bike? 'alex)))
```

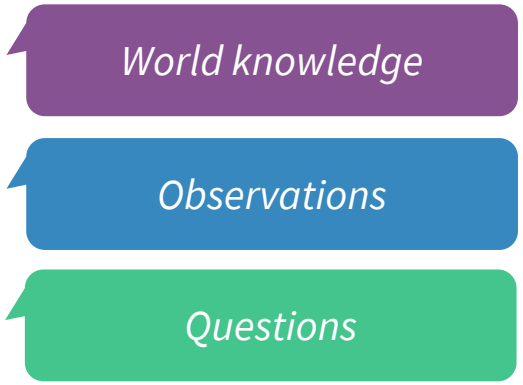
What will Alex do?

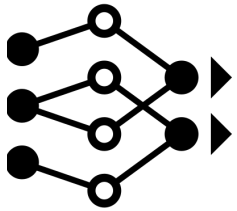
```
(query (get_actions 'alex))
```

Inference over agents using planning engine

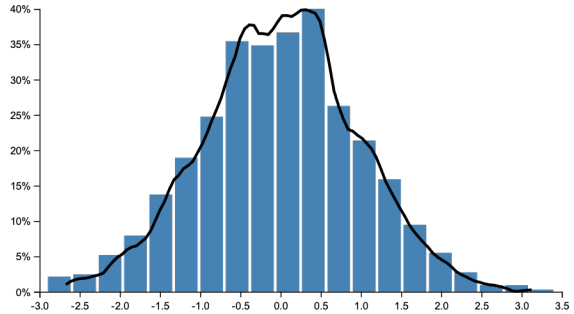
[Slides credits: Gabe Grand & Lio Wong]

Reasoning withing language using PPLs + LLMs





```
(define ...)
(condition ...)
(query ...)
```



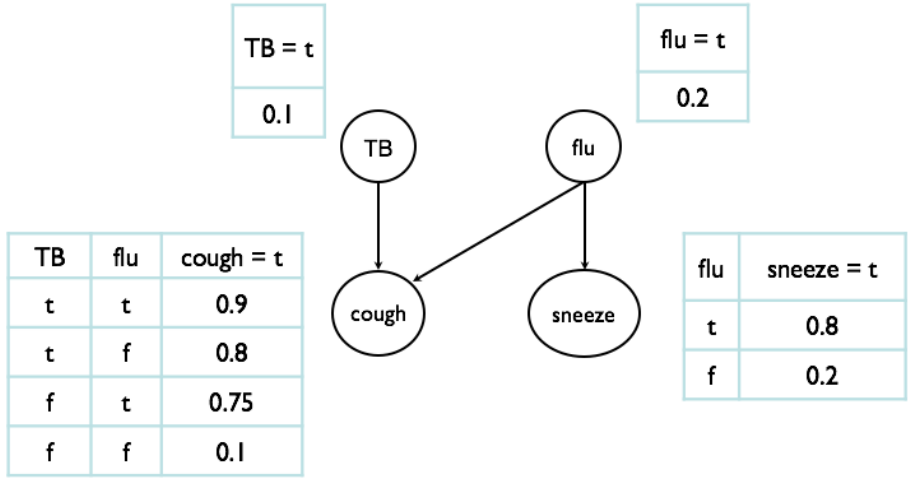
Natural language | **Meaning function** | **Probabilistic language of thought** | **Inference function** | **Distributions over possible worlds**

- A more flexible way to construct world models, conditions, and queries
- Beyond QAs (e.g., planning)
- Incorporate learning – e.g., learning new world models (and not using given world models)
- Automatic inference function selection / construction
- Connect to state-of-the-art PPLs, e.g., Gen
- …

Project ideas

Summary: Towards a probabilistic language of thought

- Bayesian networks
  - Mathematically convenient
  - Causal models
  - Hard to write code for inference algorithms



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

- Probabilistic programs
  - Have rich compositionality
    - Recall: composing programs for priors for the number game
  - Provide a complete specification of the domain (nothing hidden behind arrows)
  - Probabilistic programming languages (PPLs) provide tools for easily and flexibly implement efficient inference algorithms
  - PPLs for Bayesian models, DL frameworks (e.g., pytorch) for neural networks
  - Connected to natural language via LMs

```
var towModel = function() {
  var strength = mem(function (person) {return gaussian(50, 10)})

  var lazy = function(person) {return flip(0.1) }

  var pulling = function(person) {
    return lazy(person) ? strength(person) / 2 : strength(person) }

  var totalPulling = function (team) {return sum(map(pulling, team))}

  var winner = function (team1, team2) {
    totalPulling(team1) > totalPulling(team2) ? team1 : team2 }

  var beat = function(team1,team2){winner(team1,team2) == team1}

  condition(beat(['bob'], ['tom']))

  return strength('bob')
}
```

Why is posterior inference hard?

$$p(h|d) = \frac{p(d|h)p(h)}{Z}$$

$$Z = \int_h p(d|h)p(h)$$

Two challenges:

- Calculating Z requires integrating over all h
- Even if we had z, still hard to directly sample h w.r.t. p(h|d)

Sampling-based inference algorithms

- Monte Carlo methos:

- Rejection sampling
- Importance sampling
- Markov chain Monte Carlo (MCMC)
- Sequential Monte Carlo (SMC)

What is Monte Carlo?

- Monte Carlo is a small hillside town in Monaco (near Italy) with casino since 1865 like Los Vegas in the US. It was picked by a physicist Fermi (Italian born American) who was among the first using the sampling techniques in his effort building the first manmade nuclear reactors in 1942.

- The casino business is, literally, driven by tossing dice to simulate random events. Monte Carlo computing is to simulate samples from arbitrary probabilities by a single random function x=rand() which returns a pseudo-random number in the interval [0,1].

Sampling-based inference algorithms

- Monte Carlo methos:

- Rejection sampling
- Importance sampling
- Markov chain Monte Carlo (MCMC)
- Sequential Monte Carlo (SMC)

How to use Gen to implement sampling-based inference algorithms?