ngoodman@
stanford.edu

# Concept learning and the language of thought

Noah D. Goodman
Stanford University

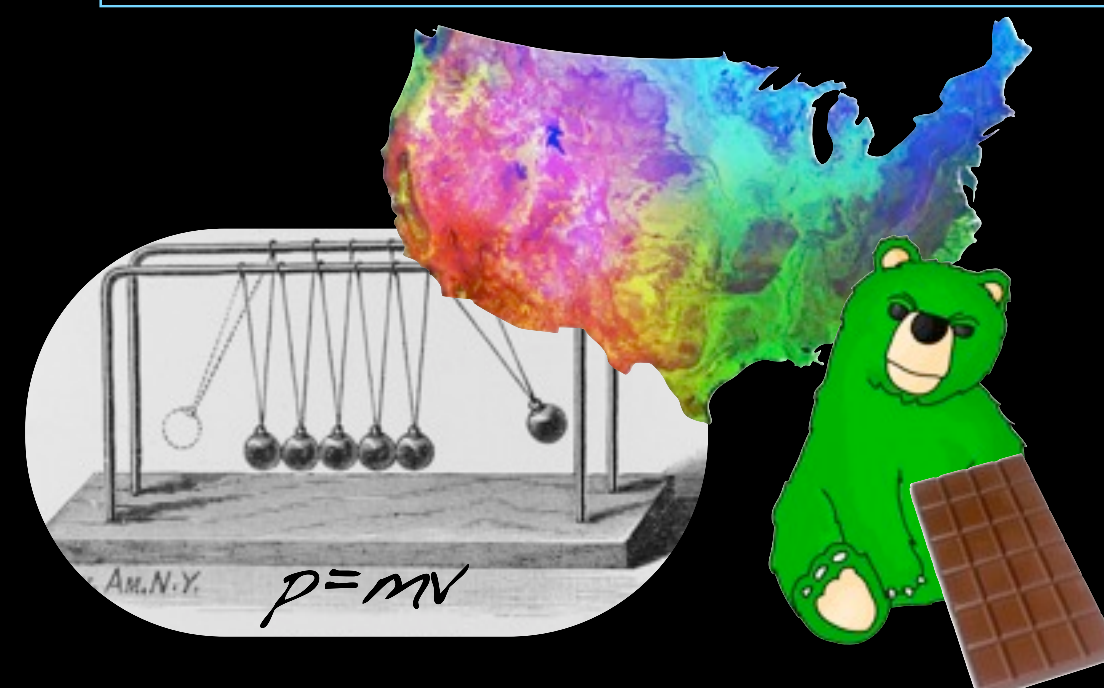
IPAM graduate summer school
July 15, 2011

# Statistics and composition

Probabilistic language of thought hypothesis

Thought is useful in an uncertain world

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


Probabilistic inference

Generative models

Compositional representations

# PLoT

- 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.

# PLoT

- The probabilistic language of thought hypothesis:
  Mental representations are functions in a **stochastic process** calculus (e.g. ψλ-calculus / Church).
  - Intuitive framework theories.
  - Flexible reasoning and language use.
  - Learning structured concepts.

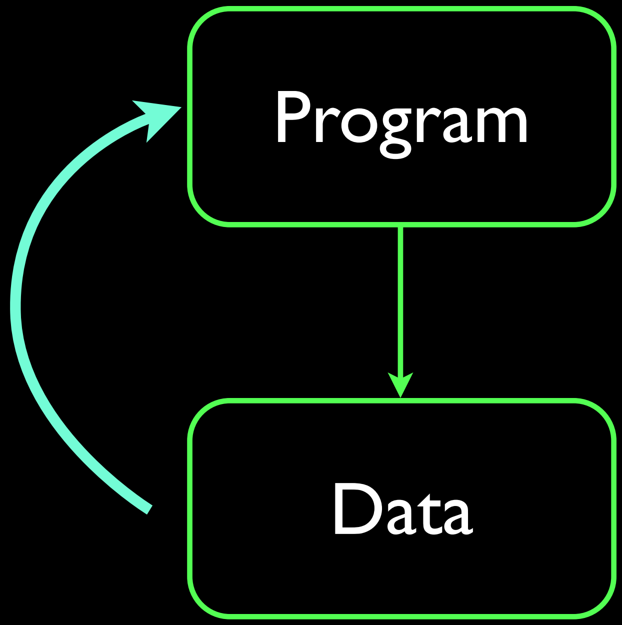
# Outline

If concepts are probabilistic programs,
then concept learning is *probabilistic program induction*.

# Outline

If concepts are probabilistic programs,
then concept learning is *probabilistic program induction*.

```
(query
 (define concept (sample-PLoT-expression))
 concept
 (and (= (noisy (sample concept)) obs1)
      (= (noisy (sample concept)) obs2)
      ...))
```



# Outline

If concepts are probabilistic programs,
then concept learning is *probabilistic program induction*.

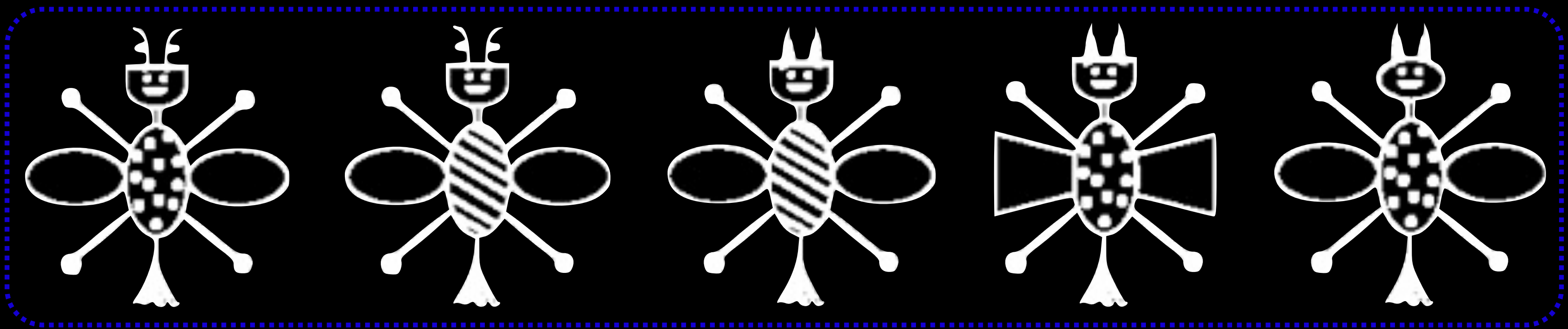
- Boolean categories
- Quantified concepts
- Natural number concepts
- Generative kinds
- Program induction

# Categorization

Medin & Schaffer (1978):

# Categorization

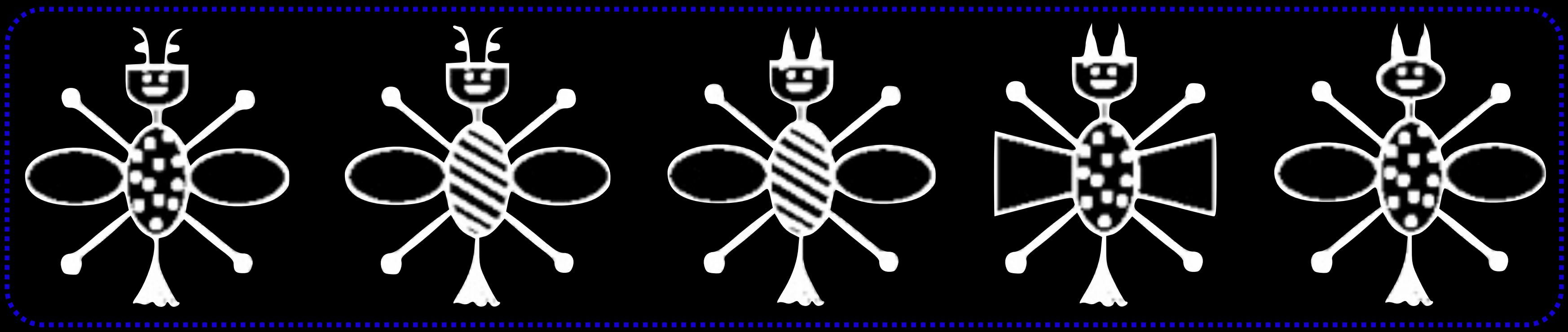
Medin & Schaffer (1978):



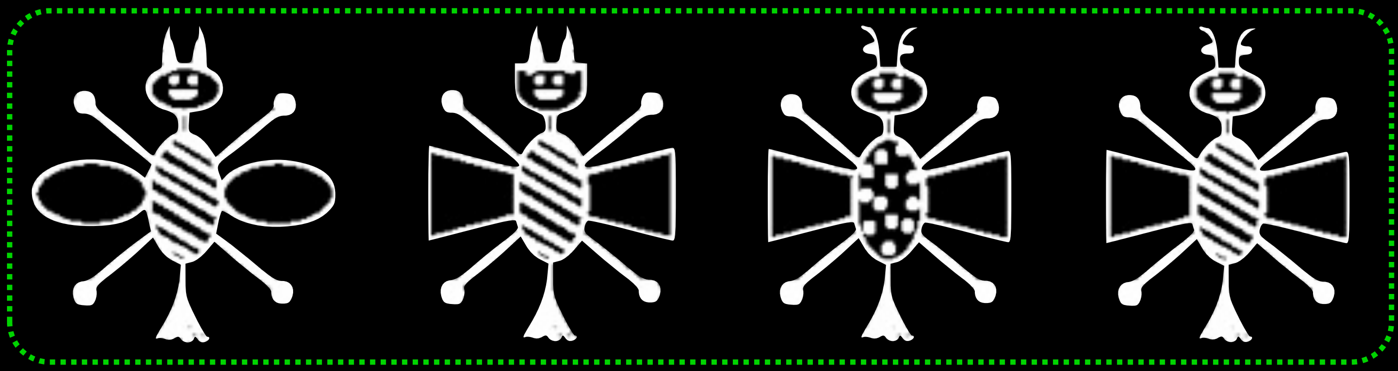
"These are **Feps**"

# Categorization

Medin & Schaffer (1978):



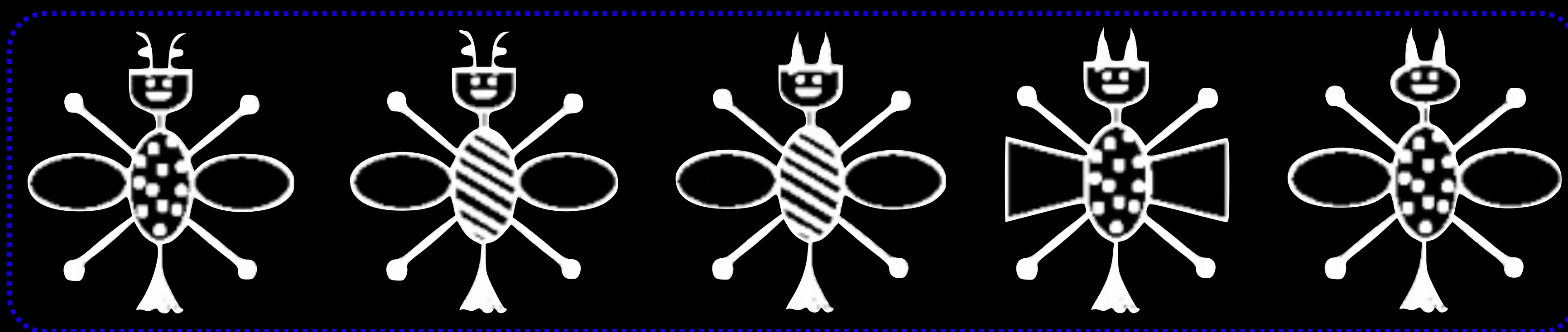
“These are **Feps**”



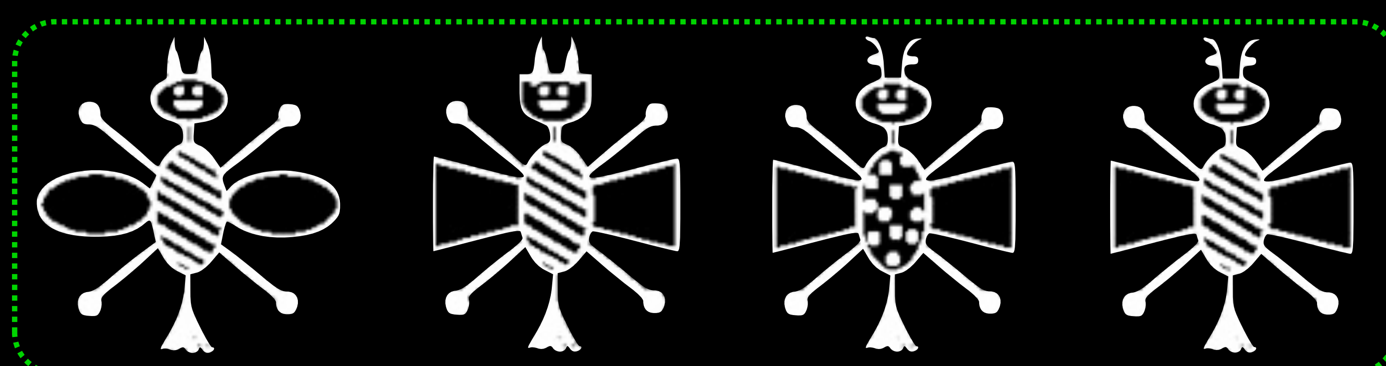
“These are **not Feps**”

# Categorization

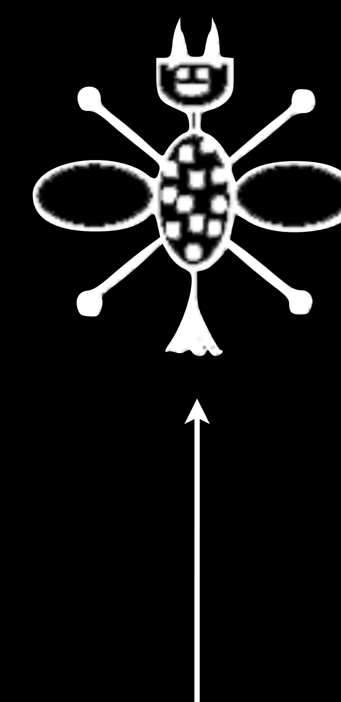
Medin & Schaffer (1978):



“These are **Feps**”



“These are **not Feps**”

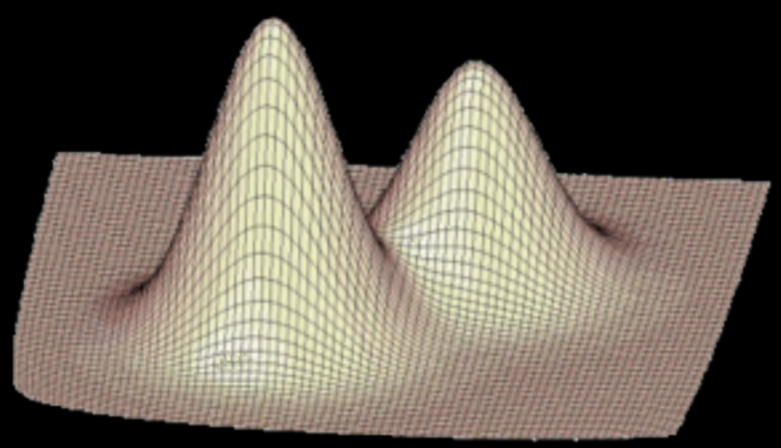


“Is this a Fep?”

# Categorization

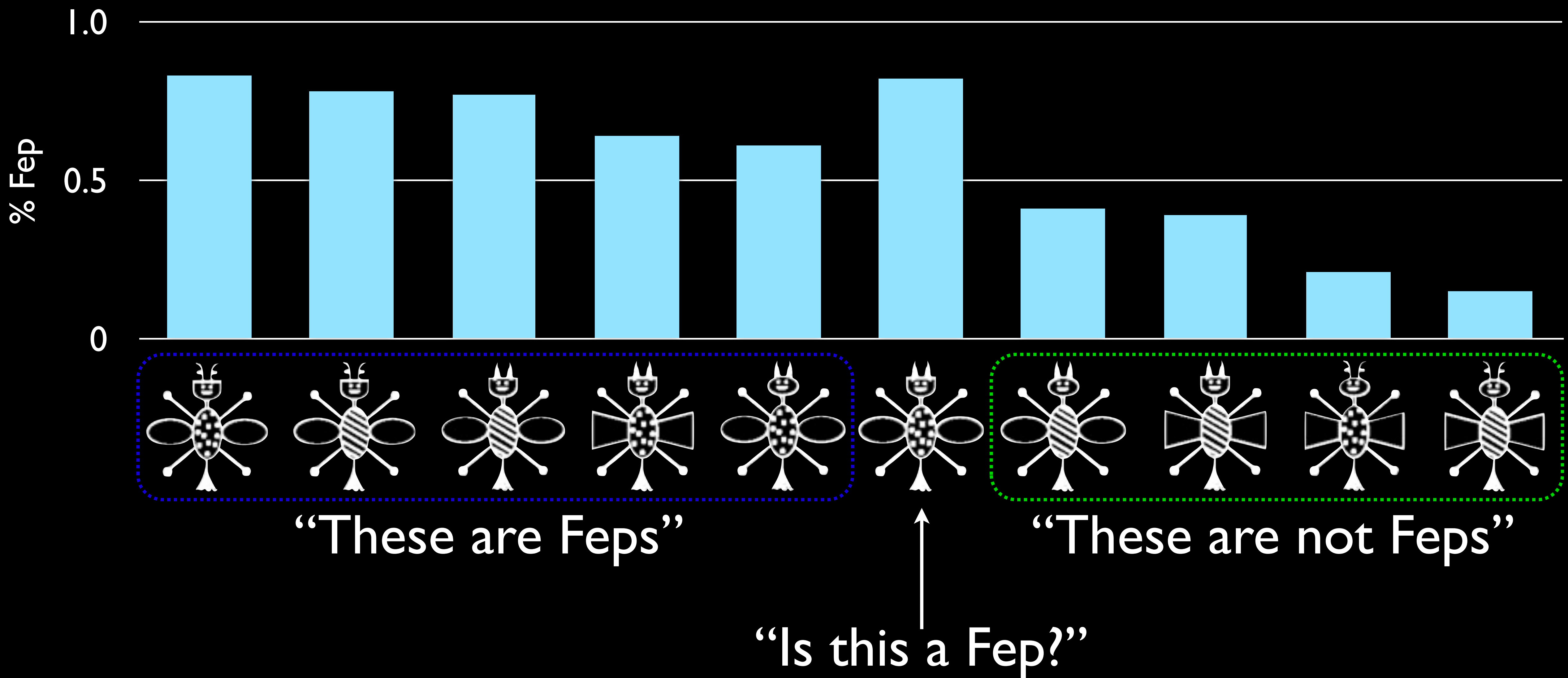
- Rule-based category learning:
  - Infinitely many concepts formed compositionally.

- Statistical category learning:
  - Graded inferences from sparse, noisy evidence.



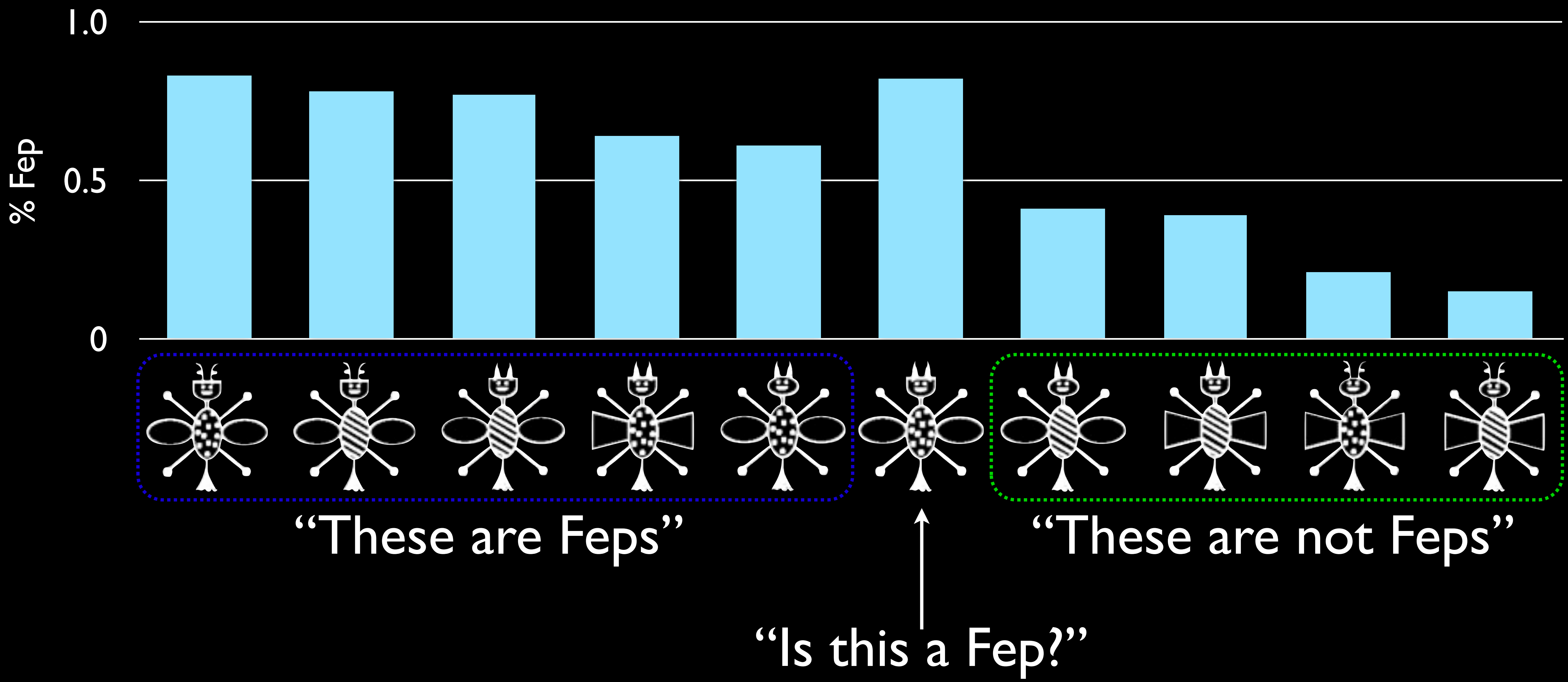
# Categorization

Medin & Schaffer, 1978 (data from Nosofsky, et al.,1994):



# Categorization

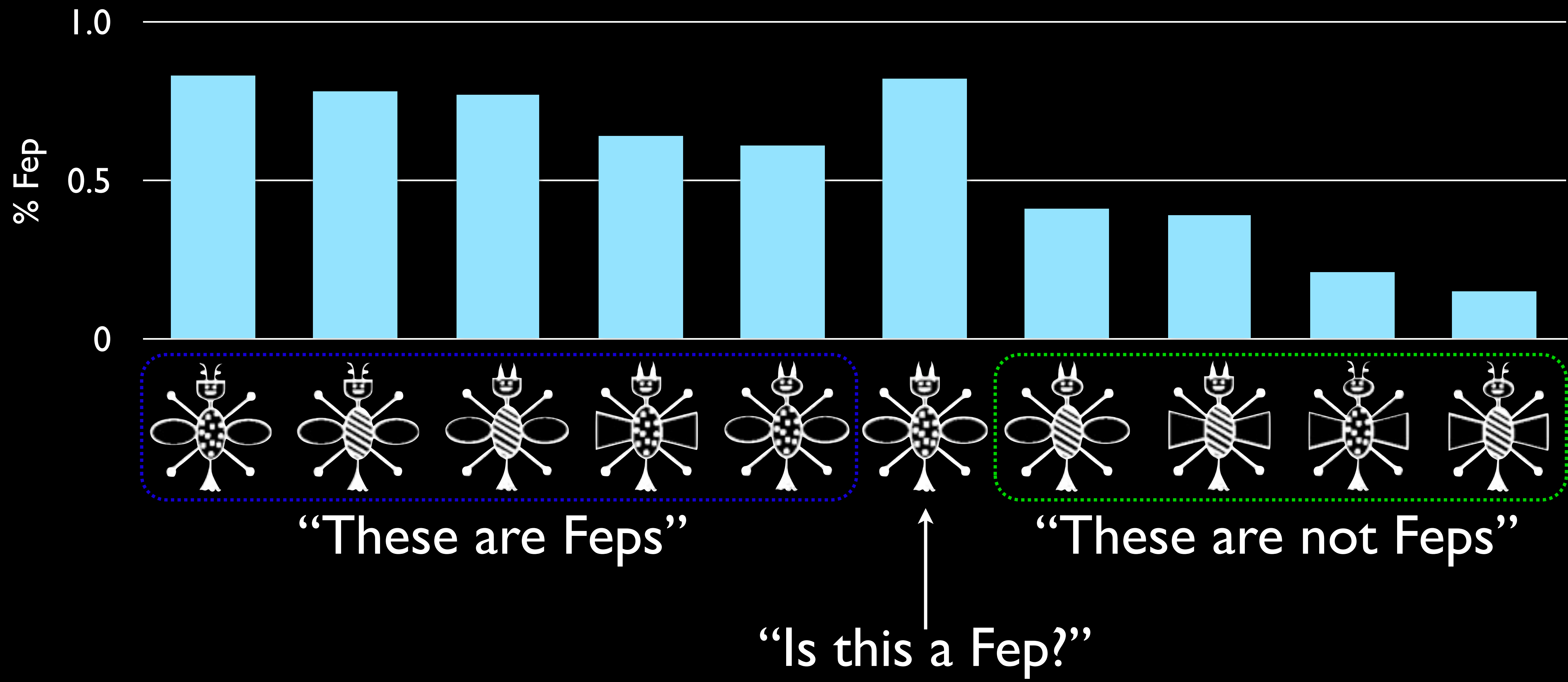
Medin & Schaffer, 1978 (data from Nosofsky, et al.,1994):




- Graded judgements

# Categorization

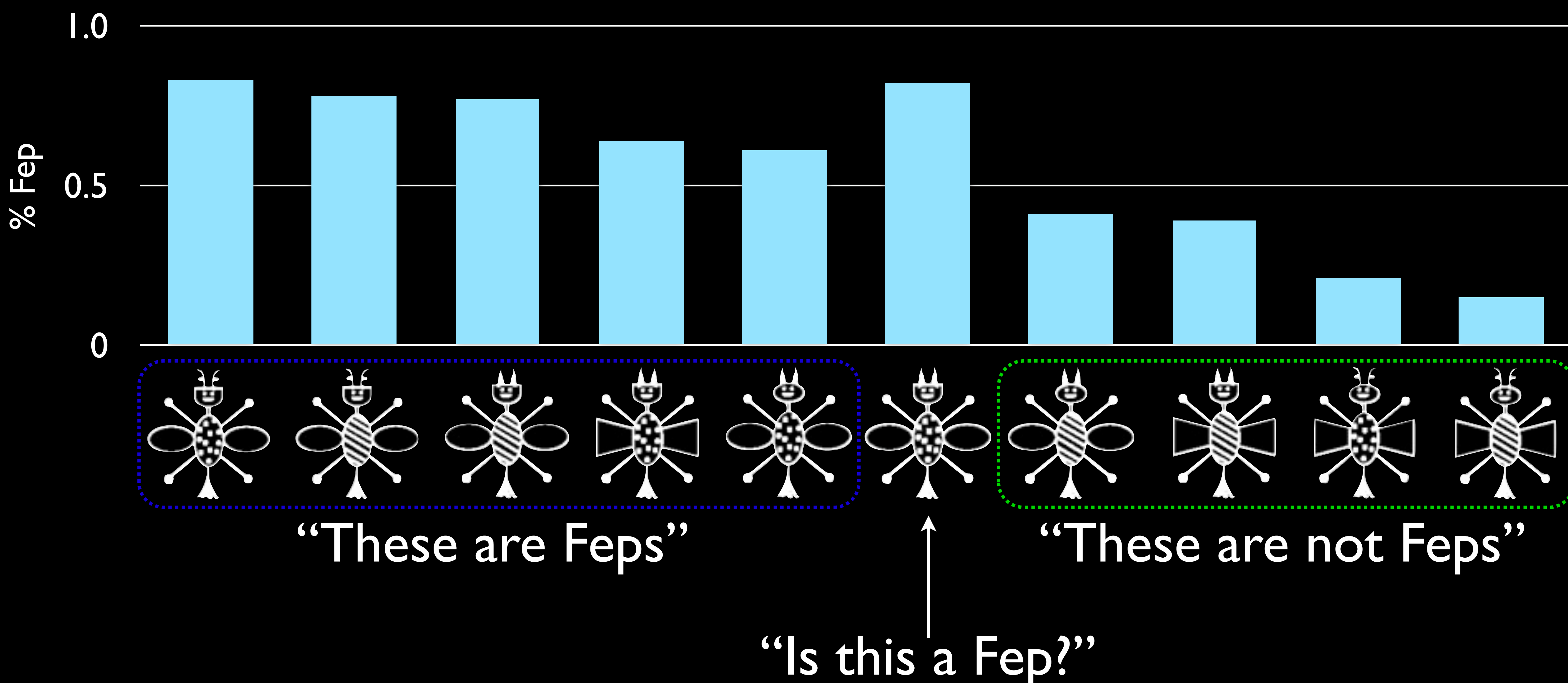
Medin & Schaffer, 1978 (data from Nosofsky, et al.,1994):




- Typicality
- Graded judgements

# Categorization

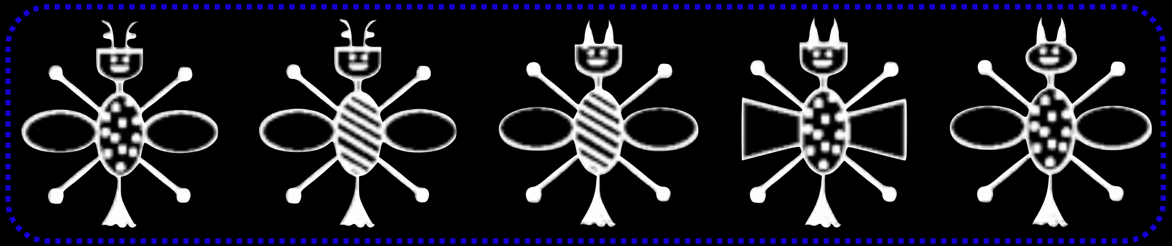
Medin & Schaffer, 1978 (data from Nosofsky, et al.,1994):

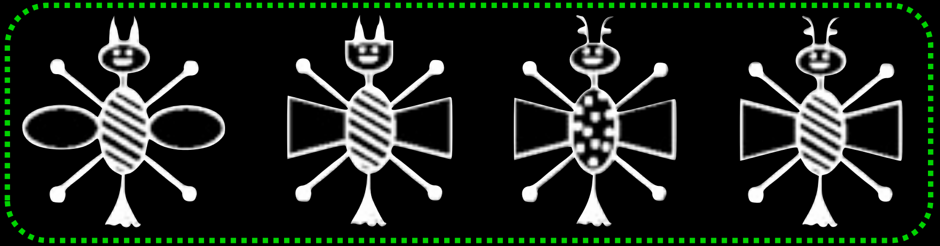



- Typicality
- Prototype enhancement
- Graded judgements

# Generating rules

“It’s a Fep if it has flat head and round wings”

Feps: 

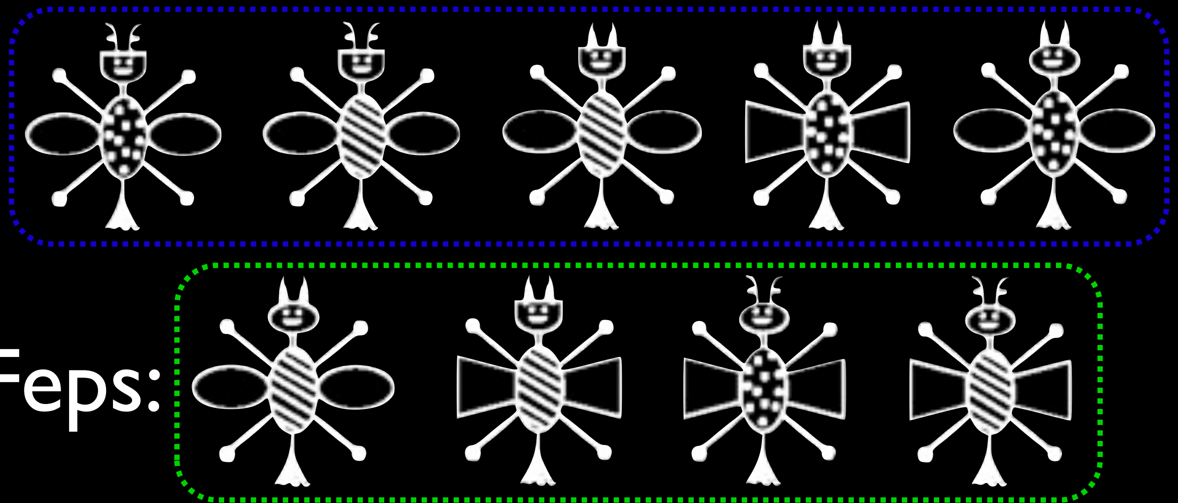
non-Feps: 

# Generating rules

```
(define fep?
 (λ (x)
  (and (flat-head x)
       (round-wings x))))
```

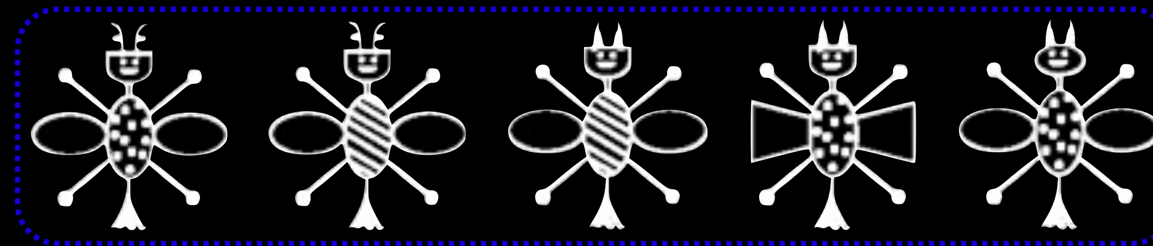
Feps:

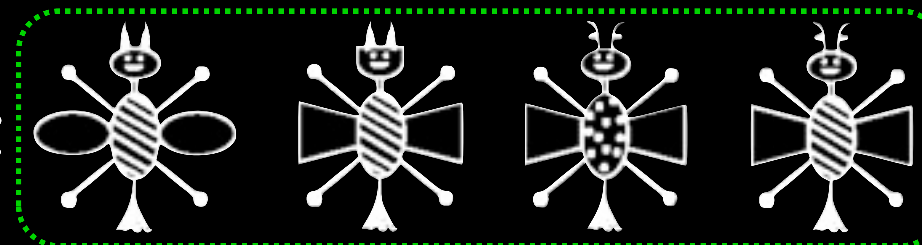
non-Feps:

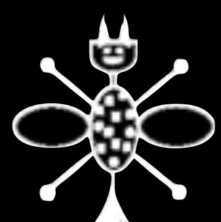


# Generating rules

```
(define fep?
 (λ (x)
  (and (flat-head x)
       (round-wings x))))
```

Feps: 

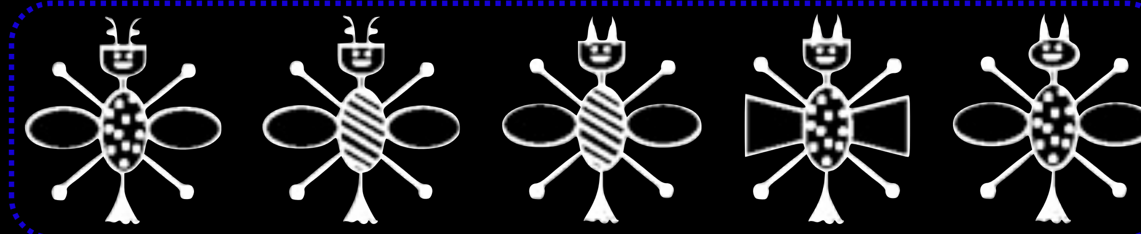
non-Feps: 

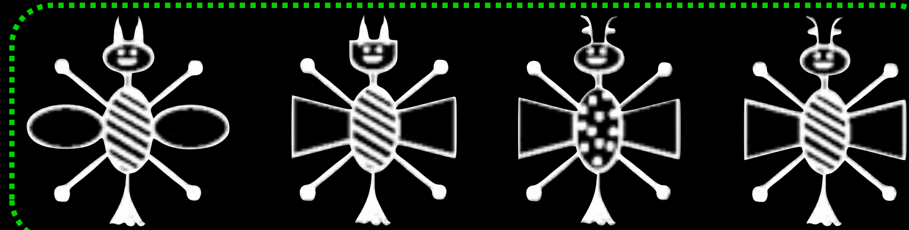
`(fep?`  `)`

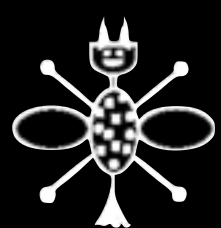
`=> true`

# Generating rules

```
(define fep?
 (λ (x)
  (and (flat-head x)
       (round-wings x))))
```

Feps: 

non-Feps: 

(fep?  )

```
 => true
```

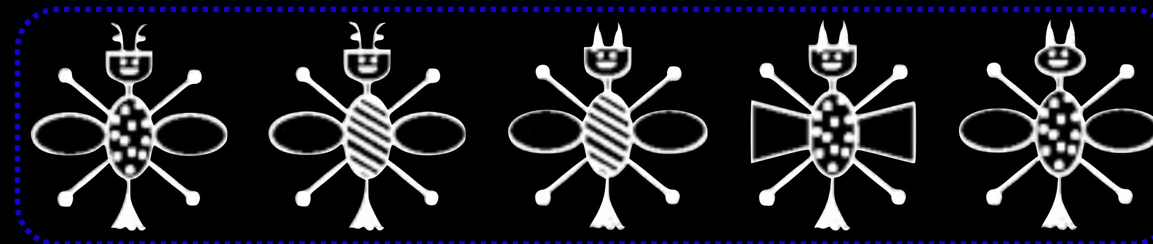
```
(define rule-generator
 (λ ()
  (if (flip 0.3)
      (sample-feature)
      (combine-rules (sample-feature)
                     (rule-generator)))
```

```
(define combine-rules
 (λ (r1 r2)
  (λ (x) (and (r1 x) (r2 x)))
```

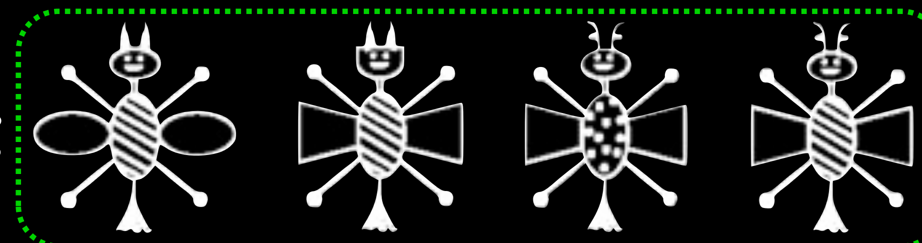
# Generating rules

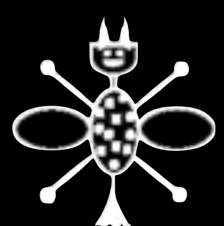
```
(define fep?
 (λ (x)
  (and (flat-head x)
       (round-wings x))))
```

Feps:



non-Feps:



(fep?  )

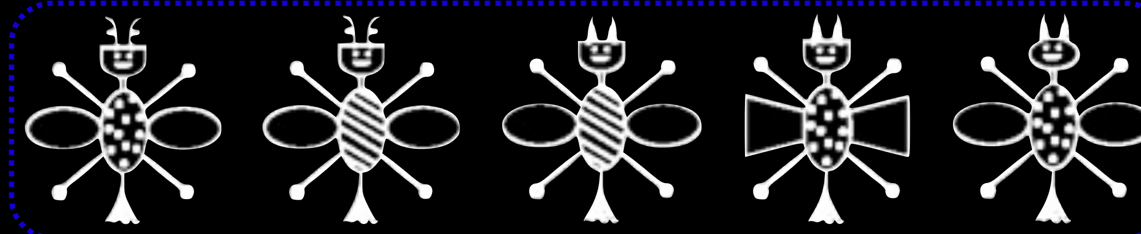
=> true

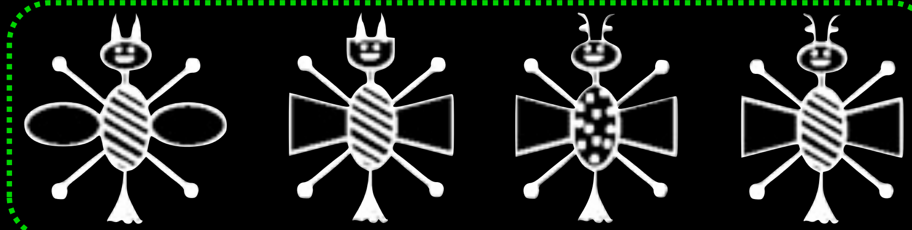
```
(define rule-generator
 (λ ()
  (if (flip 0.3)
      (sample-feature)
      (combine-rules (sample-feature)
                     (rule-generator)))
```

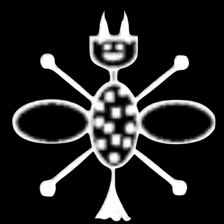
```
(define combine-rules
 (λ (r1 r2)
   (λ (x) (and (r1 x) (r2 x)))
```

# Generating rules

```
(define fep?
 (λ (x)
  (and (flat-head x)
       (round-wings x))))
```

Feps: 

non-Feps: 

(fep?  )

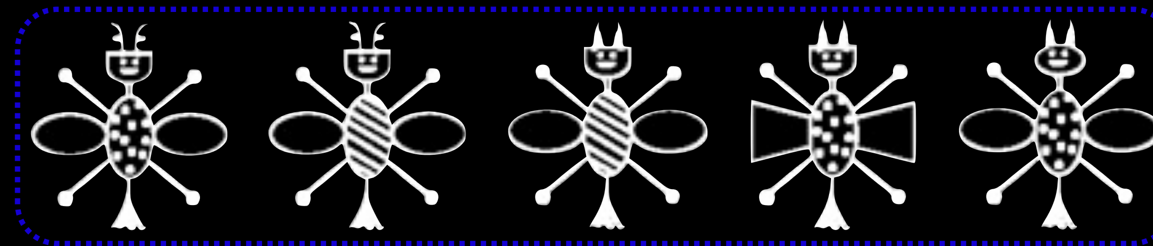
=> true

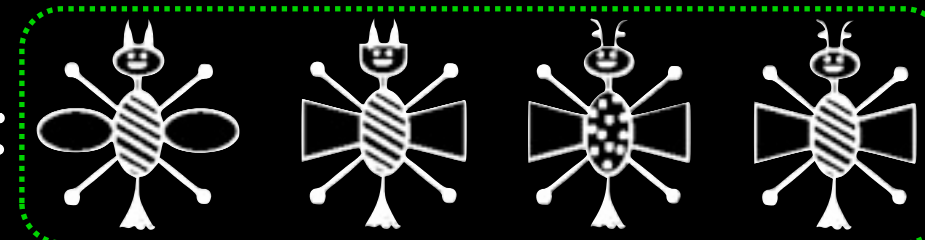
```
(define rule-generator
 (λ ()
  (if (flip 0.3)
      (sample-feature)
      (combine-rules (sample-feature)
                     (rule-generator)))))
```

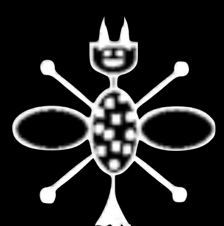
```
(define combine-rules
 (λ (r1 r2)
  (λ (x) (and (r1 x) (r2 x)))))
```

# Generating rules

```
(define fep?
 (λ (x)
  (and (flat-head x)
       (round-wings x))))
```

Feps: 

non-Feps: 

(fep?  )

```
 => true
```

```
(define rule-generator
 (λ ()
  (if (flip 0.3)
      (sample-feature)
      (combine-rules (sample-feature)
                     (rule-generator)))))
```

```
(define combine-rules
 (λ (r1 r2)
  (λ (x) (and (r1 x) (r2 x)))))
```

- Longer rules have lower probability (Occam’s razor).

# Generating rules

Put uncertainty over rule probabilities:

```
(define rule-prob (uniform 0 1))
(define rule-generator
 (λ ()
  (if (flip rule-prob)
      ...
```

Generate disjunctive normal form (DNF) rules:

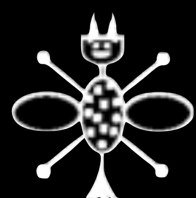
```
(define fep?
 (λ (x)
  (and (or (flat-head x) ...)
       (or (round-wings x) ...)
       ...)))
```

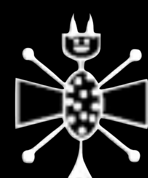
The general idea:
grammar-based induction.

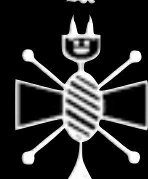
# Inducing rules

Inference:

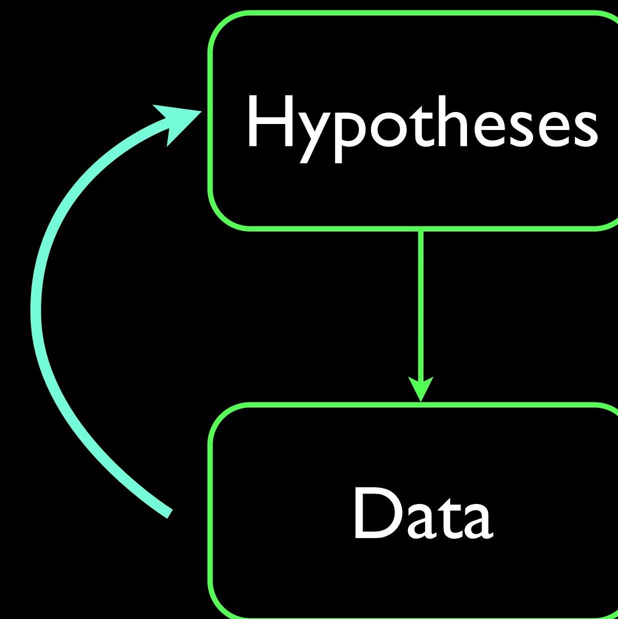
```
(query
 (define rule (rule-generator))
```

(rule  )

(and (= (rule  ) true)

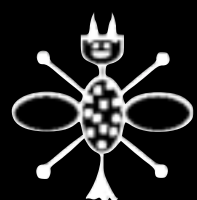
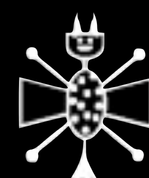
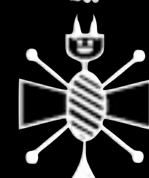
(= (rule  ) false)

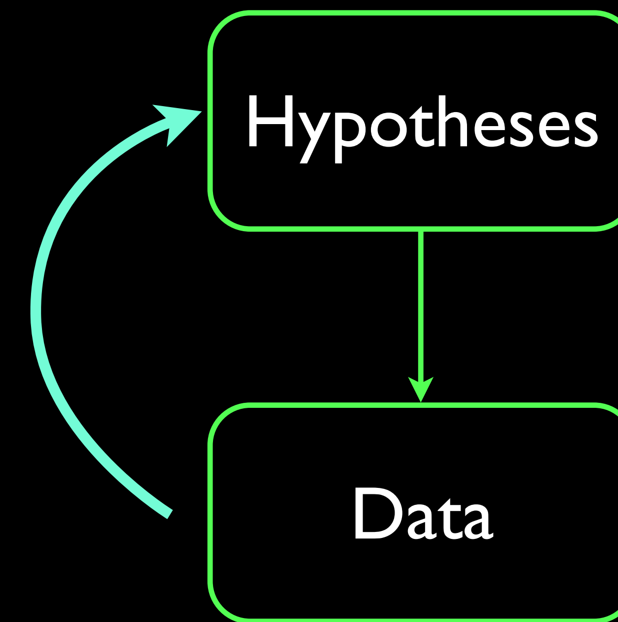
```
      ...))
```



# Inducing rules

## Inference:

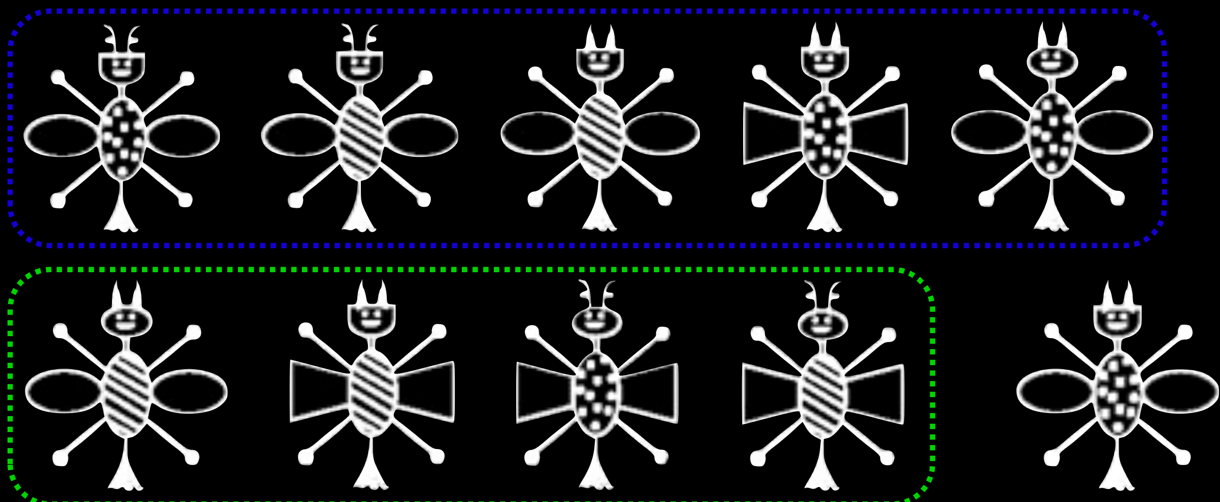
`(query`
` (define rule (rule-generator))`
` (rule`  `)`
` (and (= (noisy (rule`  `)) true)`
`      (= (noisy (rule`  `)) false)`
`      ...))`

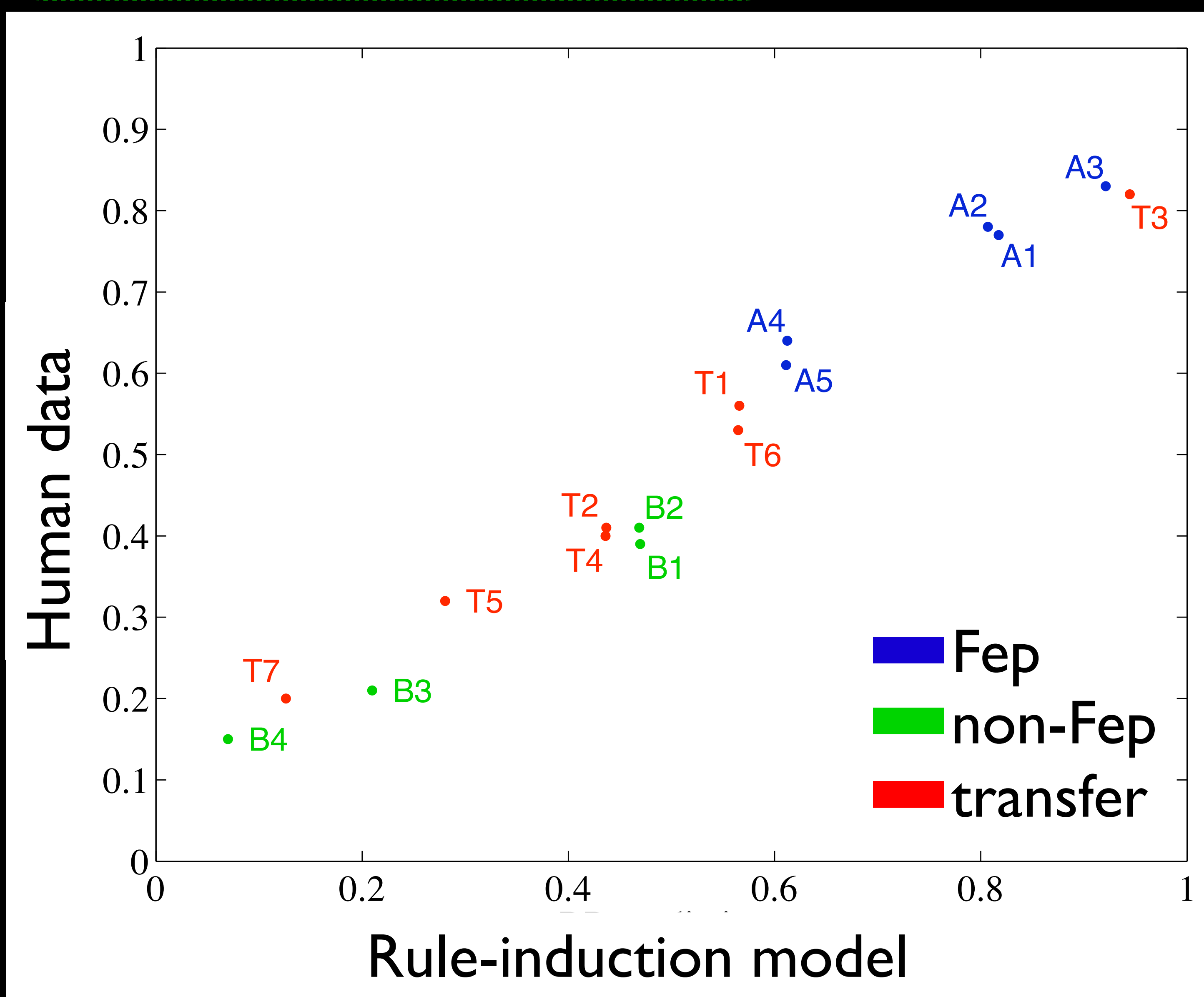



## Observation noise:

```
(define noisy
 (λ (bit) (if (flip b) bit (not bit))))
```

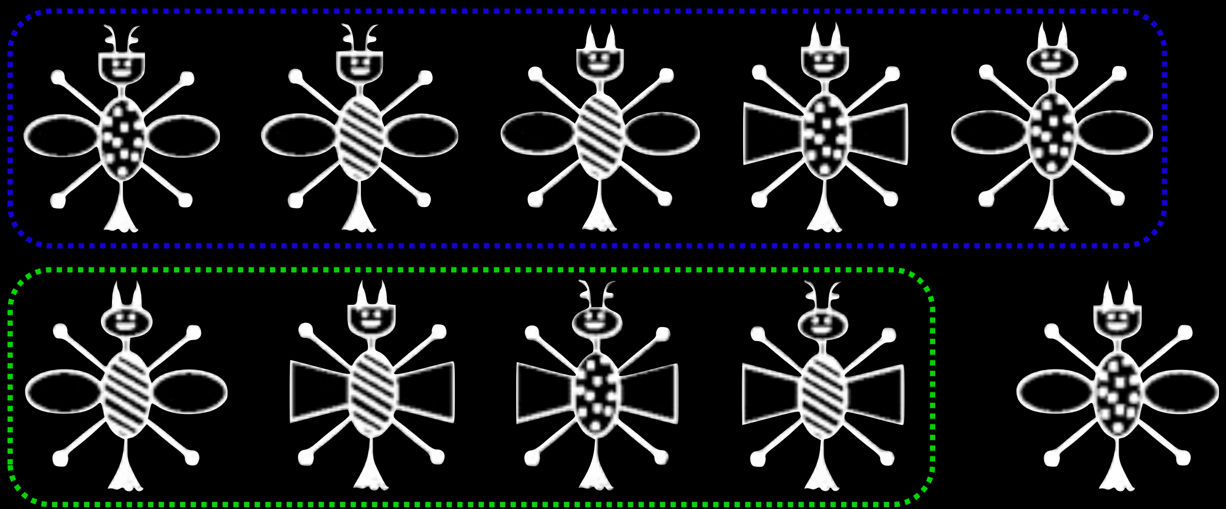
# Categorization

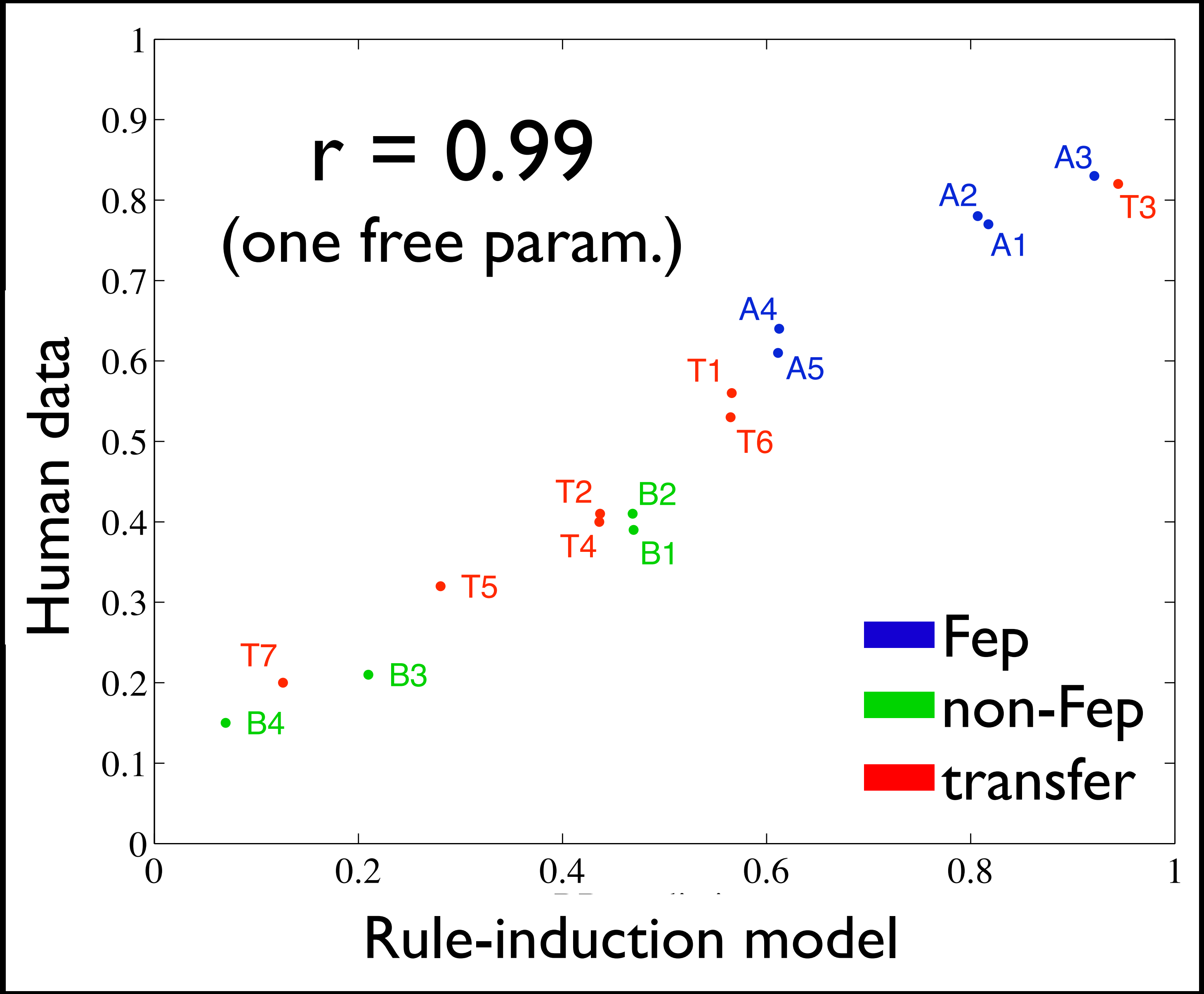





Goodman, et al.
(2007, 2008a, 2008b)

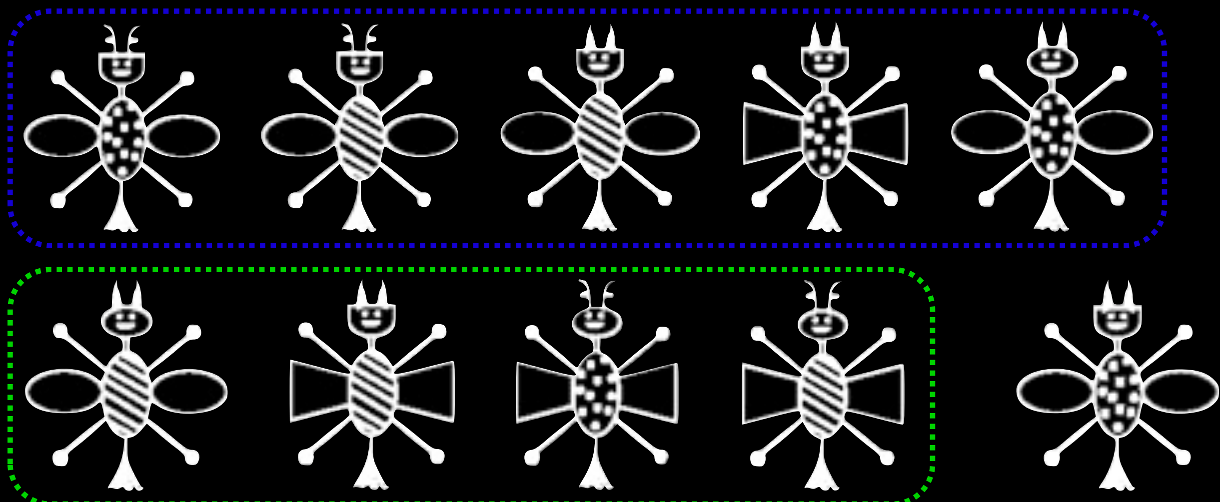
# Categorization

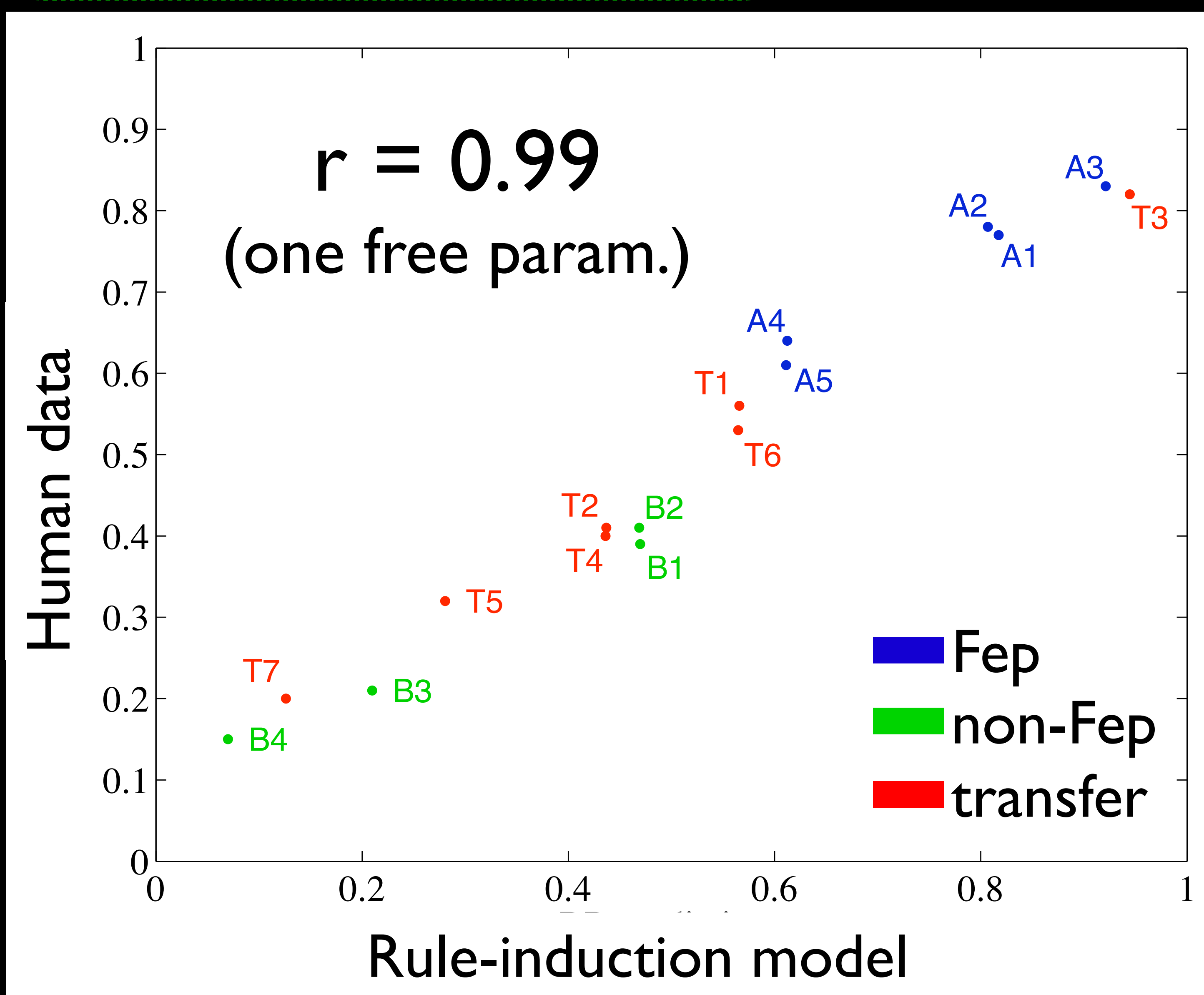





Goodman, et al.
(2007, 2008a, 2008b)

# Categorization

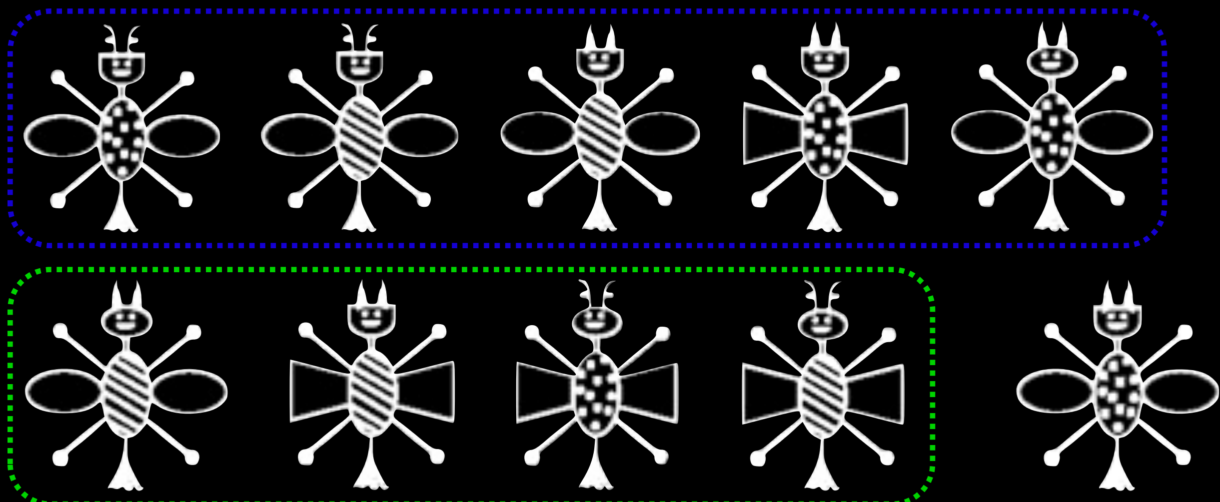


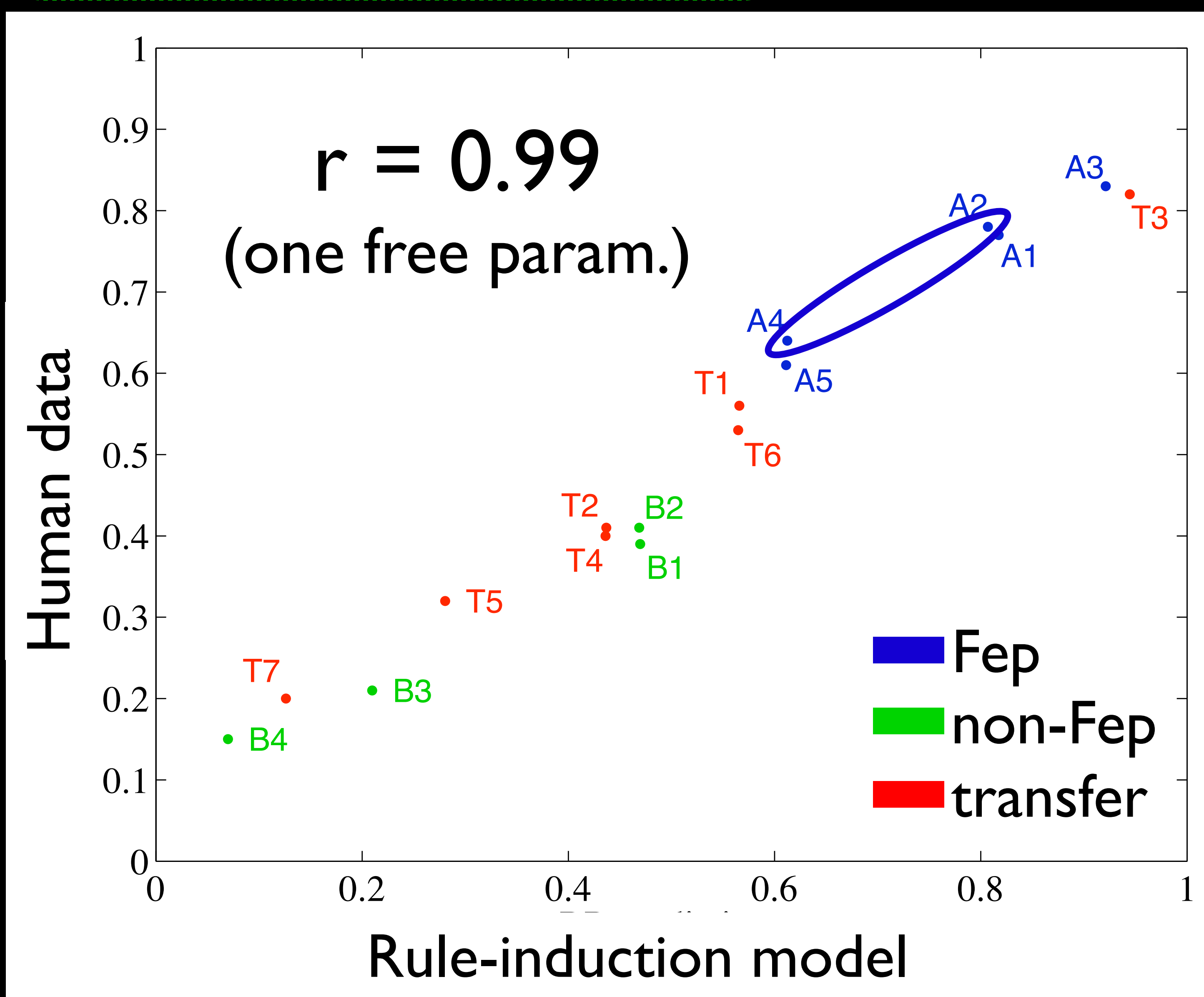



- Graded judgments

Goodman, et al.
(2007, 2008a, 2008b)

# Categorization

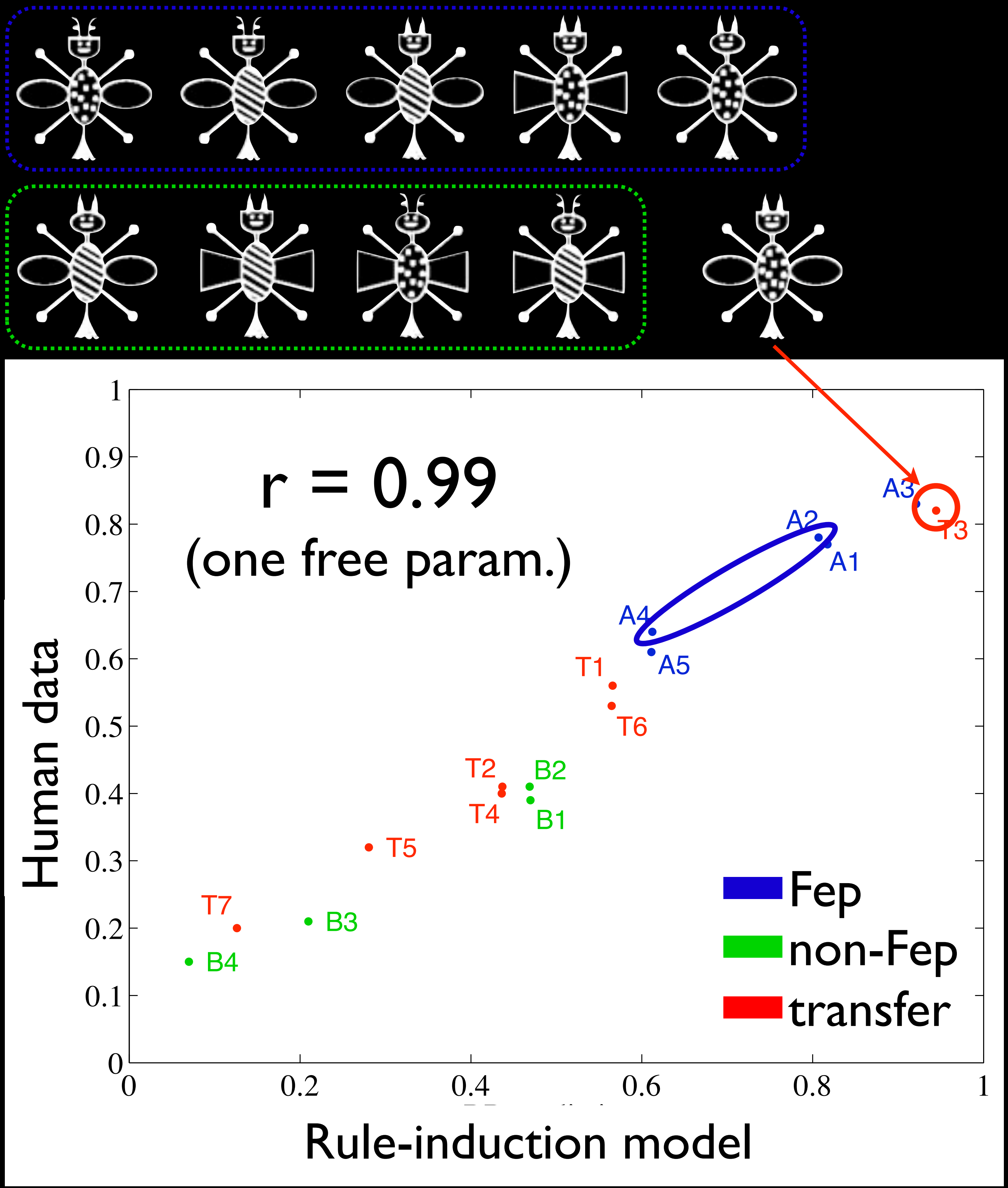





- Graded judgments
- Typicality

Goodman, et al.
(2007, 2008a, 2008b)

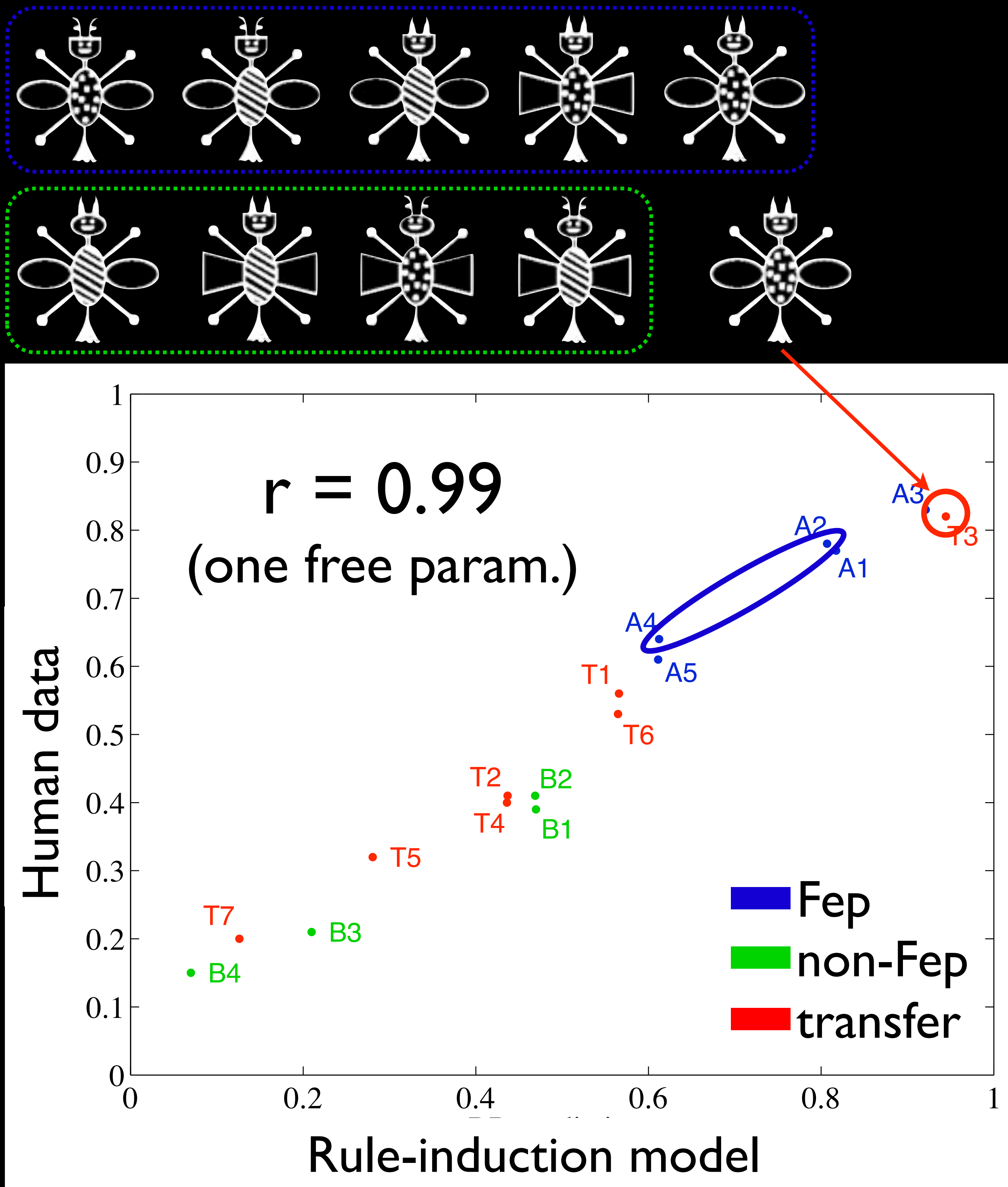
# Categorization




- Graded judgments
- Typicality
- Prototype enhancement

Goodman, et al.
(2007, 2008a, 2008b)

# Categorization

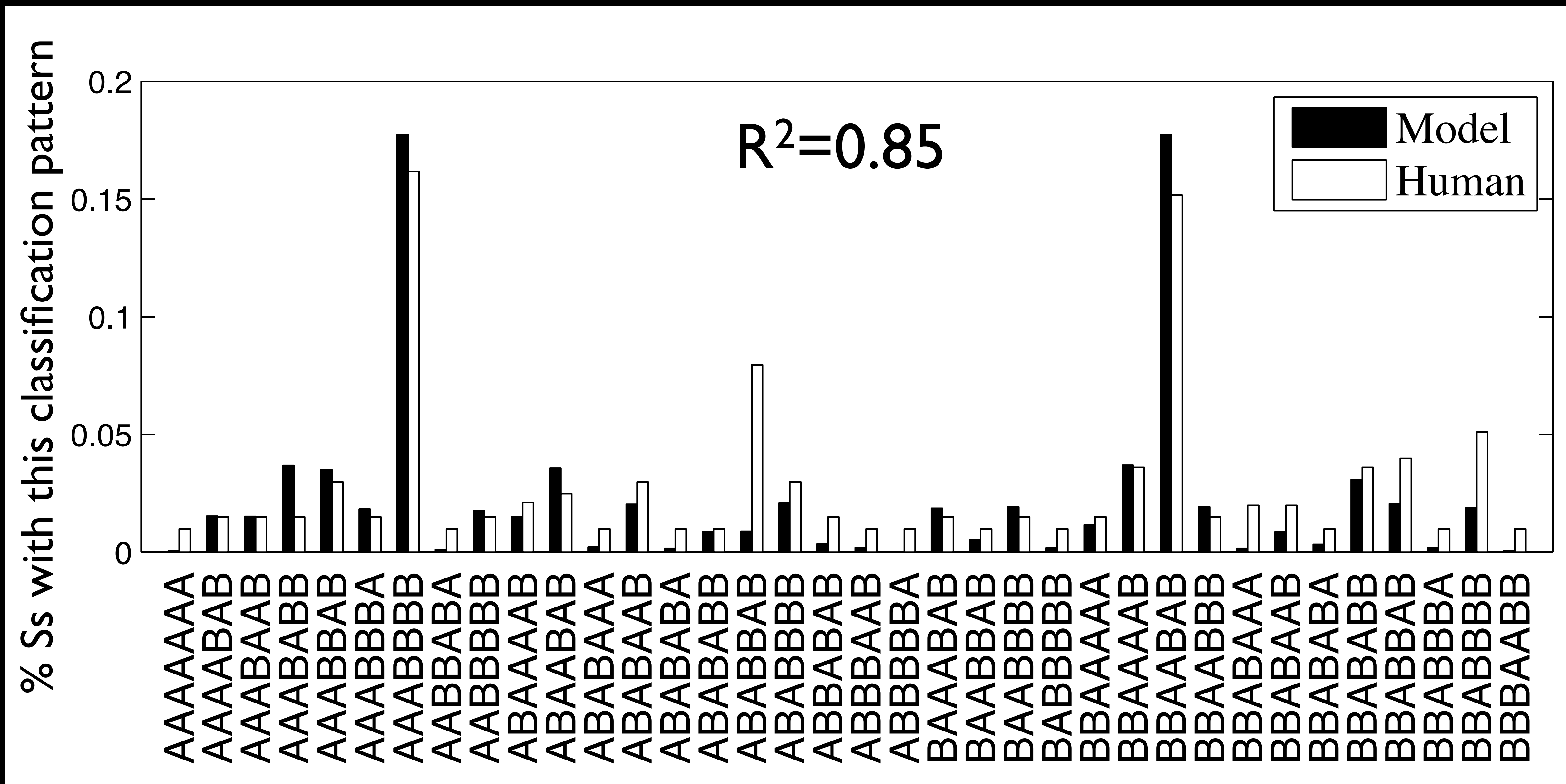



- Graded judgments
- Typicality
- Prototype enhancement
- Selective attention

Goodman, et al.
(2007, 2008a, 2008b)

# Categorization

Individual generalization patterns (for 7 transfer items):




- Model assumes individuals **sample** a (few) rule(s).

# Complexity shift

- With more exposure to training examples subjects shift from simple to complex categorization patterns. (Medin, et al., 1982)
- Tradeoff between observation noise and simplicity bias (Occam's razor).

# Complexity shift

- With more exposure to training examples subjects shift from simple to complex categorization patterns. (Medin, et al., 1982)
- Tradeoff be observatio simplicity bias (Occam's razor).

Single feature:

```
(define fep?
 (λ (x) (flat-head x)))
```

# Complexity shift

- With more exposure to training examples subjects shift from simple to complex categorization patterns. (Medin, et al., 1982)
- Tradeoff be observatio simplicity bias (Occam's razor).

Single feature:

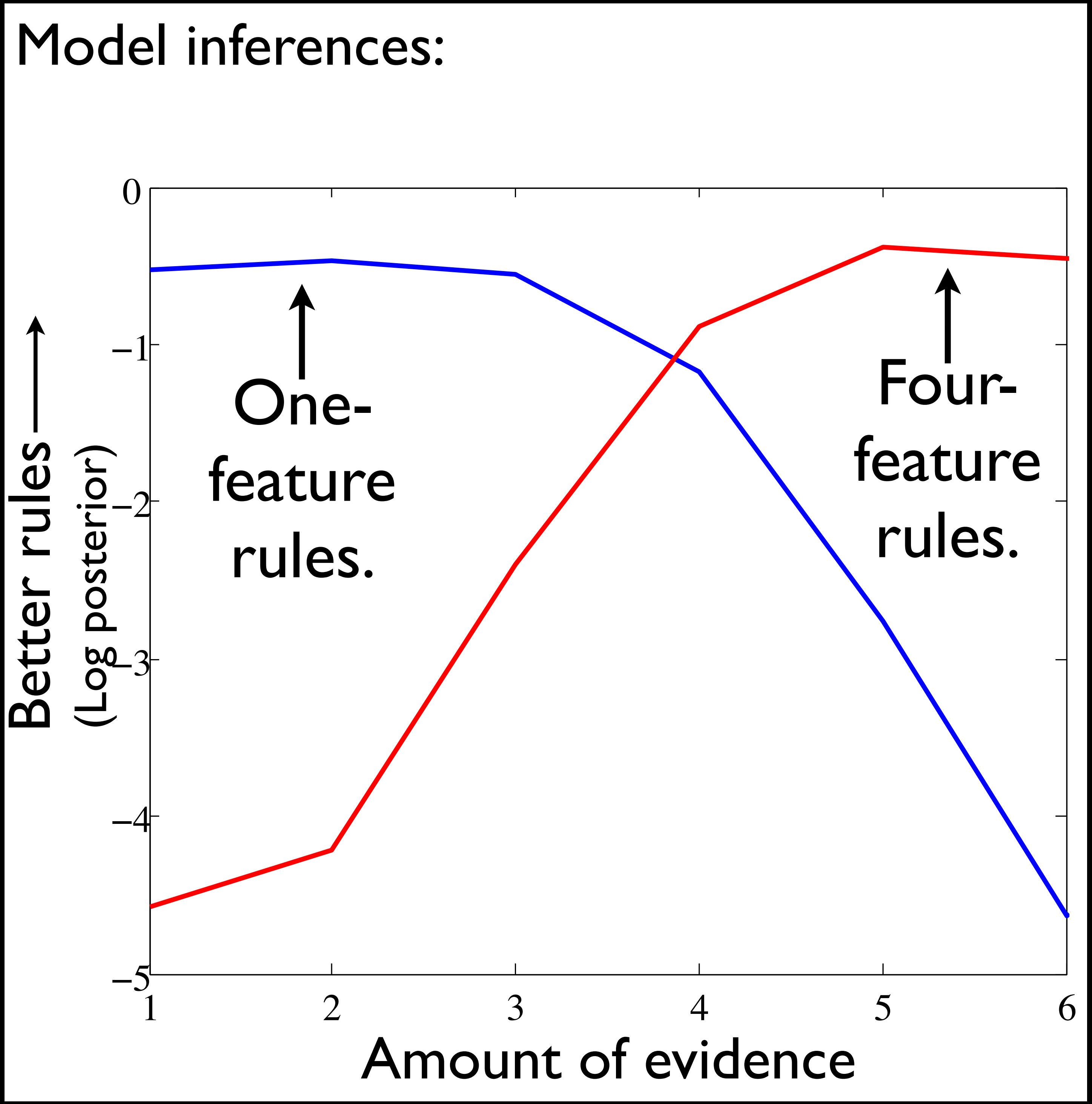
```
(define fep?
 (λ (x) (flat-head x)))
```

Multiple feature:

```
(define fep?
 (λ (x) (or (and (flat-head x)
                 (round-wings x))
            (and (round-head x)
                 (square-wings x)))))
```

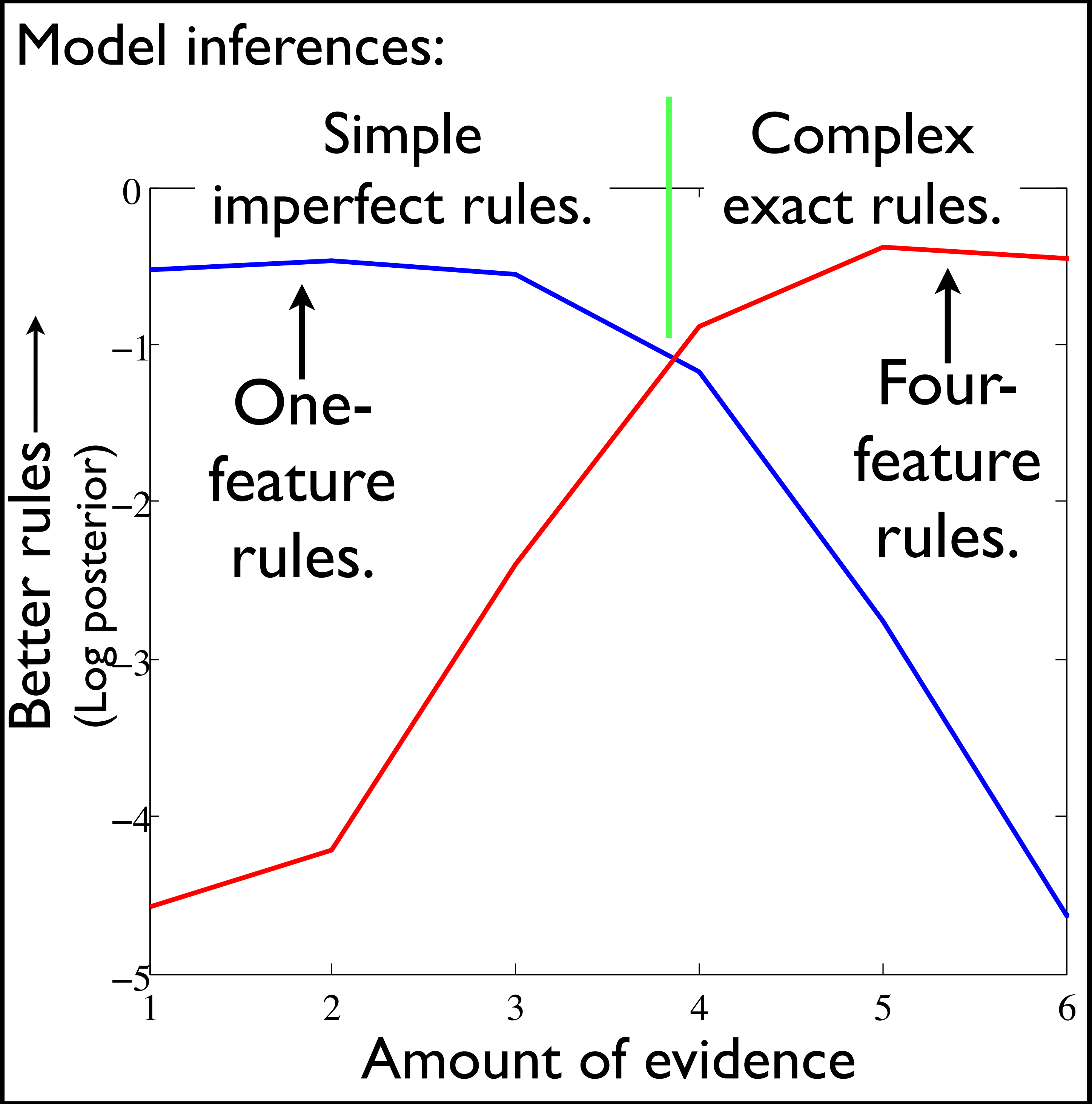
# Complexity shift

- With more exposure to training examples subjects shift from simple to complex categorization patterns. (Medin, et al., 1982)
- Tradeoff between observation noise and simplicity bias (Occam's razor).



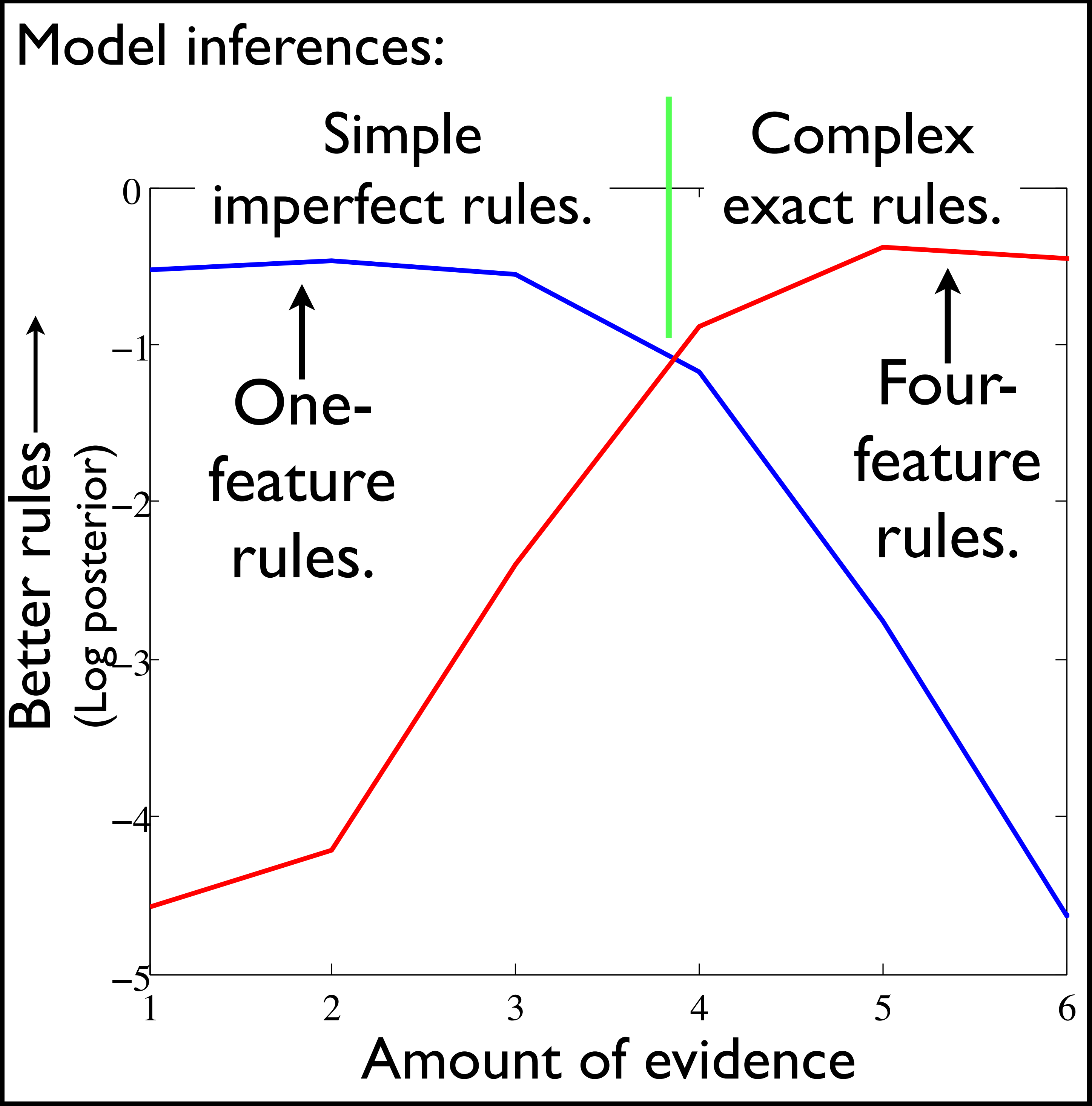
# Complexity shift

- With more exposure to training examples subjects shift from simple to complex categorization patterns. (Medin, et al., 1982)
- Tradeoff between observation noise and simplicity bias (Occam's razor).



# Complexity shift

- With more exposure to training examples subjects shift from simple to complex categorization patterns. (Medin, et al., 1982)
- Tradeoff between observation noise and simplicity bias (Occam's razor).

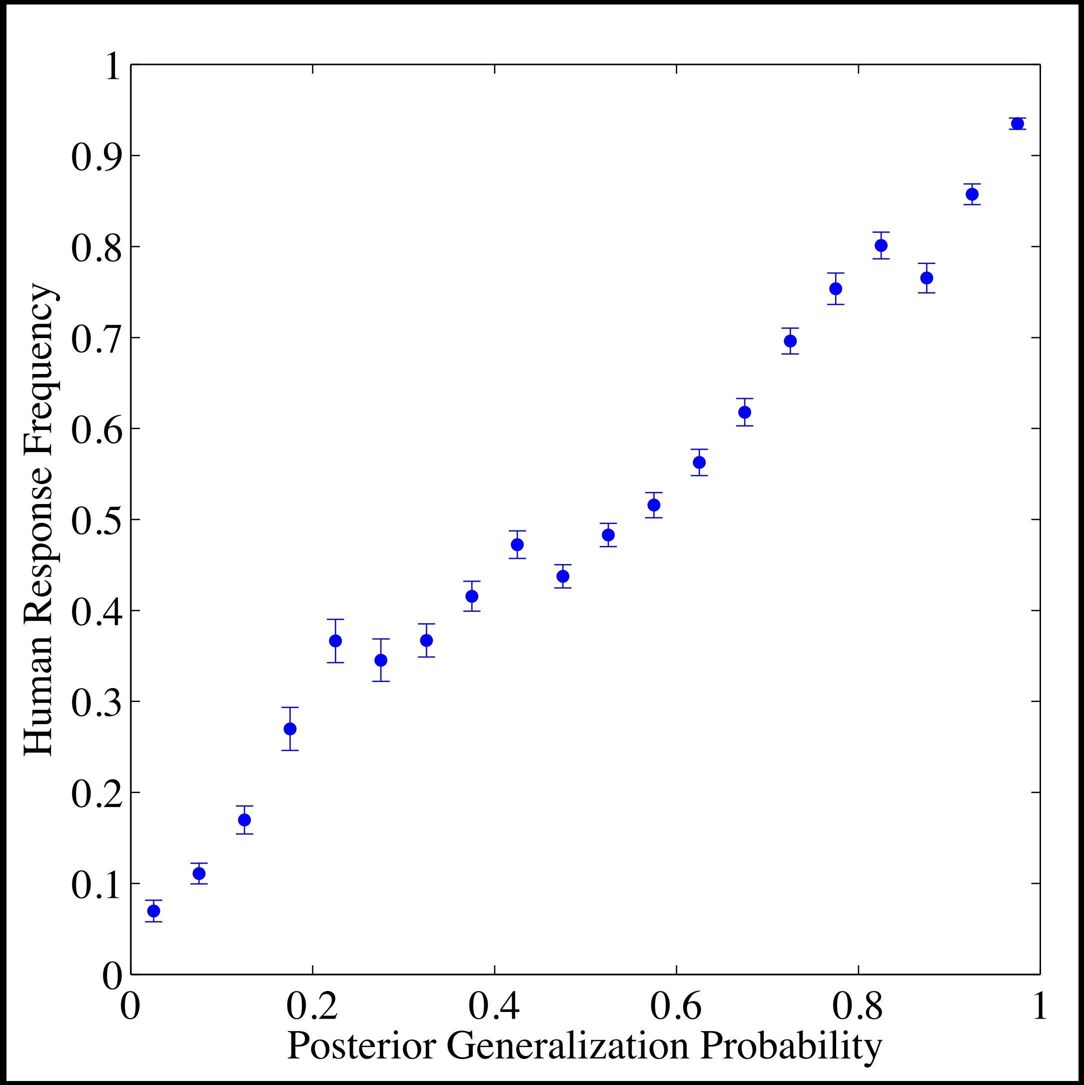


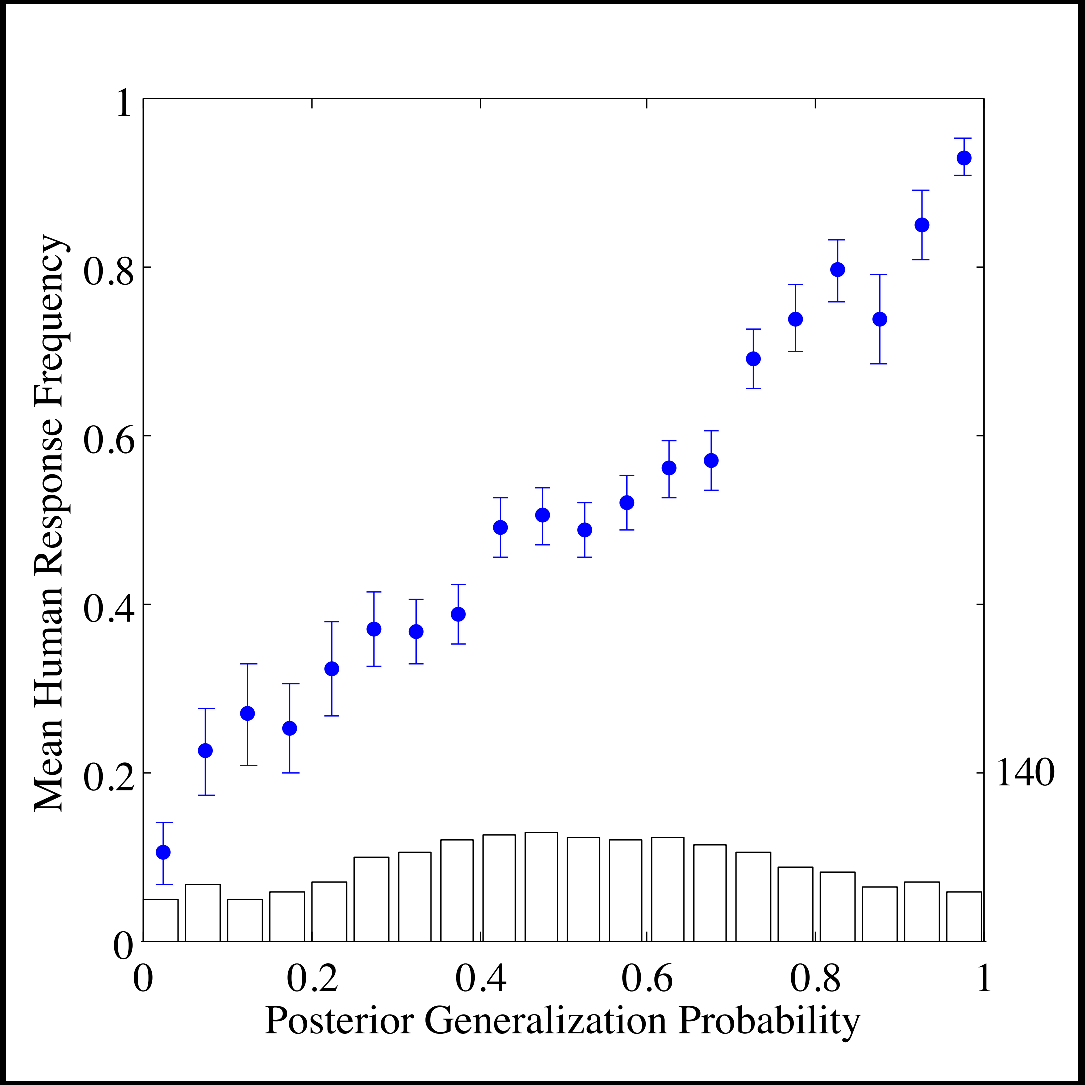
# Evaluating languages

- Induction to the language generated by the DNF grammar explains important phenomena (and fits relevant data).
- But is this the *right* LoT?
  - Test on wider data set?
  - Compare to other propositional languages?

# Broader test

- 7 Boolean features.
- 43 randomly generated concepts (3-6 pos. + 2 neg. exs)
- 128 judgements (~122 transfer questions)



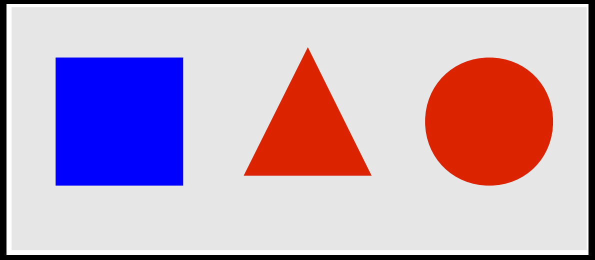

Goodman, et al (2008)

# Evaluating languages

- High-throughput MTurk experiment.
  - 108 concepts,
    - Boolean (*circle or red*)
    - Context-dependent (“Determiners”)
      (*unique largest* , *exists another with same shape*)
  - 2 orders per concept,
  - 1596 participants.

Piantadosi, Goodman, Tenenbaum (in prep)

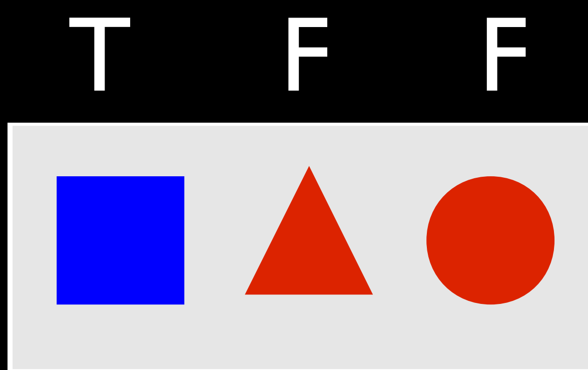
# Evaluating languages



- High-throughput MTurk experiment.
  - 108 concepts,
    - Boolean (*circle or red*)
    - Context-dependent ("Determiners")
      (*unique largest , exists another with same shape*)
  - 2 orders per concept,
  - 1596 participants.

Piantadosi, Goodman, Tenenbaum (in prep)

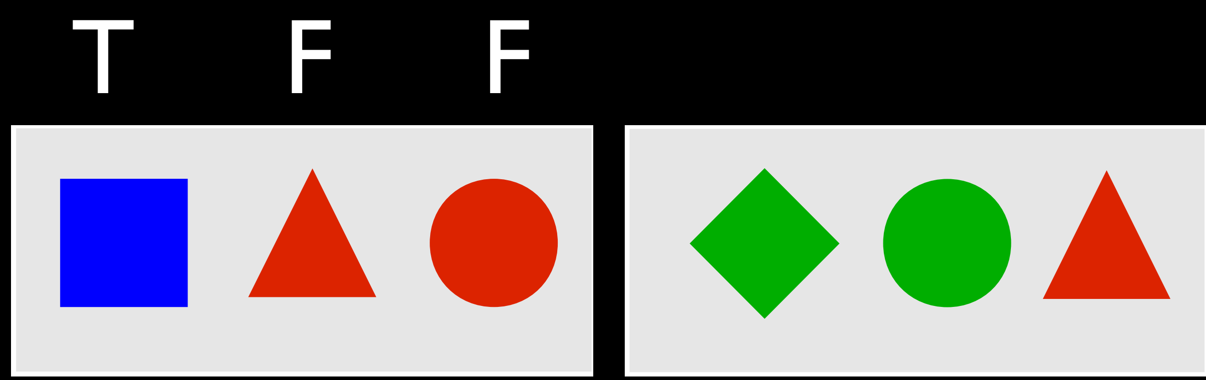
# Evaluating languages




- High-throughput MTurk experiment.
  - 108 concepts,
    - Boolean (*circle or red*)
    - Context-dependent ("Determiners")
      (*unique largest* , *exists another with same shape*)
  - 2 orders per concept,
  - 1596 participants.

Piantadosi, Goodman, Tenenbaum (in prep)

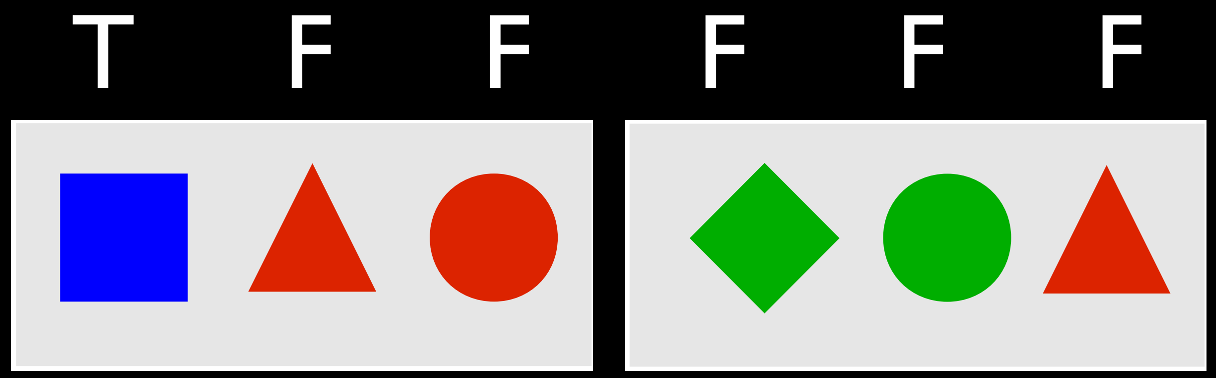
# Evaluating languages




- High-throughput MTurk experiment.
  - 108 concepts,
    - Boolean (*circle or red*)
    - Context-dependent ("Determiners")
      (*unique largest* , *exists another with same shape*)
  - 2 orders per concept,
  - 1596 participants.

Piantadosi, Goodman, Tenenbaum (in prep)

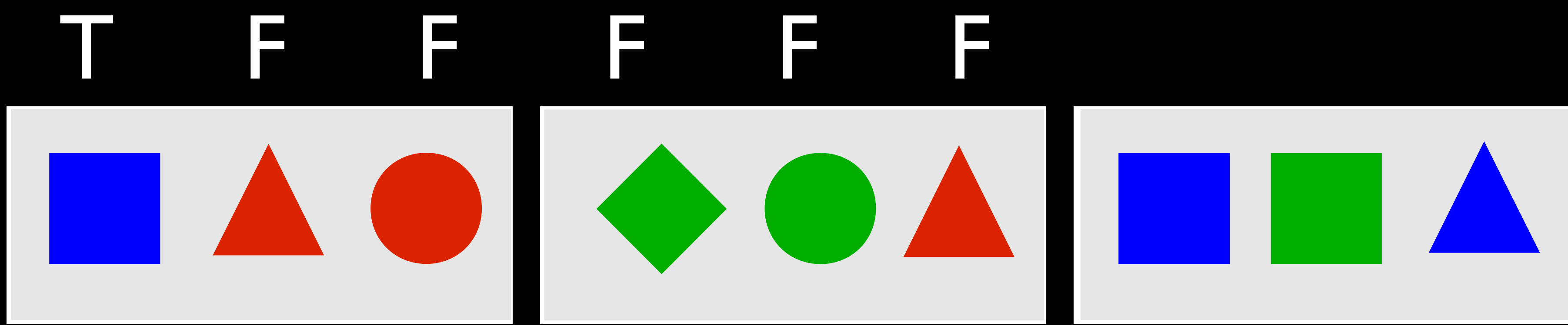
# Evaluating languages




- High-throughput MTurk experiment.
  - 108 concepts,
    - Boolean (*circle or red*)
    - Context-dependent ("Determiners") (*unique largest* , *exists another with same shape*)
  - 2 orders per concept,
  - 1596 participants.

Piantadosi, Goodman, Tenenbaum (in prep)

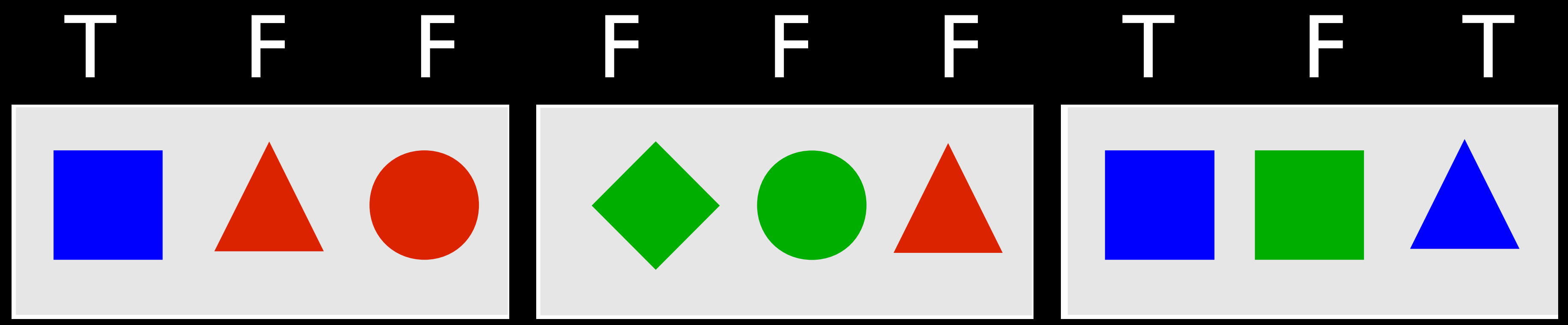
# Evaluating languages




- High-throughput MTurk experiment.
  - 108 concepts,
    - Boolean (*circle or red*)
    - Context-dependent (“Determiners”)
      (*unique largest* , *exists another with same shape*)
  - 2 orders per concept,
  - 1596 participants.

Piantadosi, Goodman, Tenenbaum (in prep)

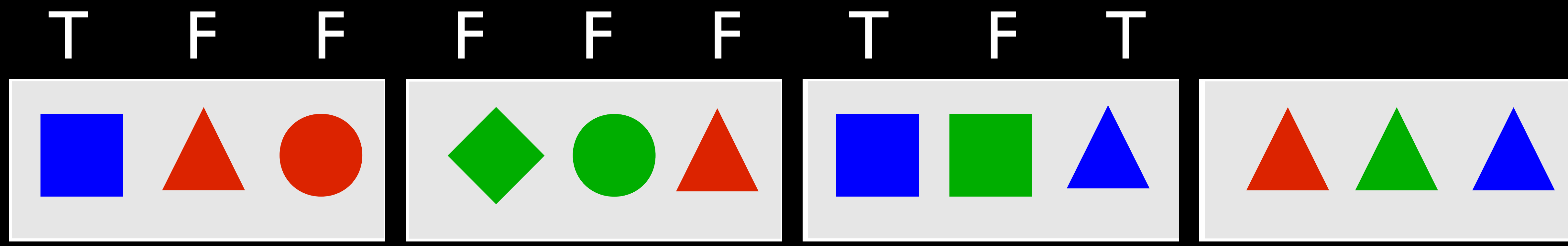
# Evaluating languages




- High-throughput MTurk experiment.
  - 108 concepts,
    - Boolean (*circle or red*)
    - Context-dependent (“Determiners”)
      (*unique largest* , *exists another with same shape*)
  - 2 orders per concept,
  - 1596 participants.

Piantadosi, Goodman, Tenenbaum (in prep)

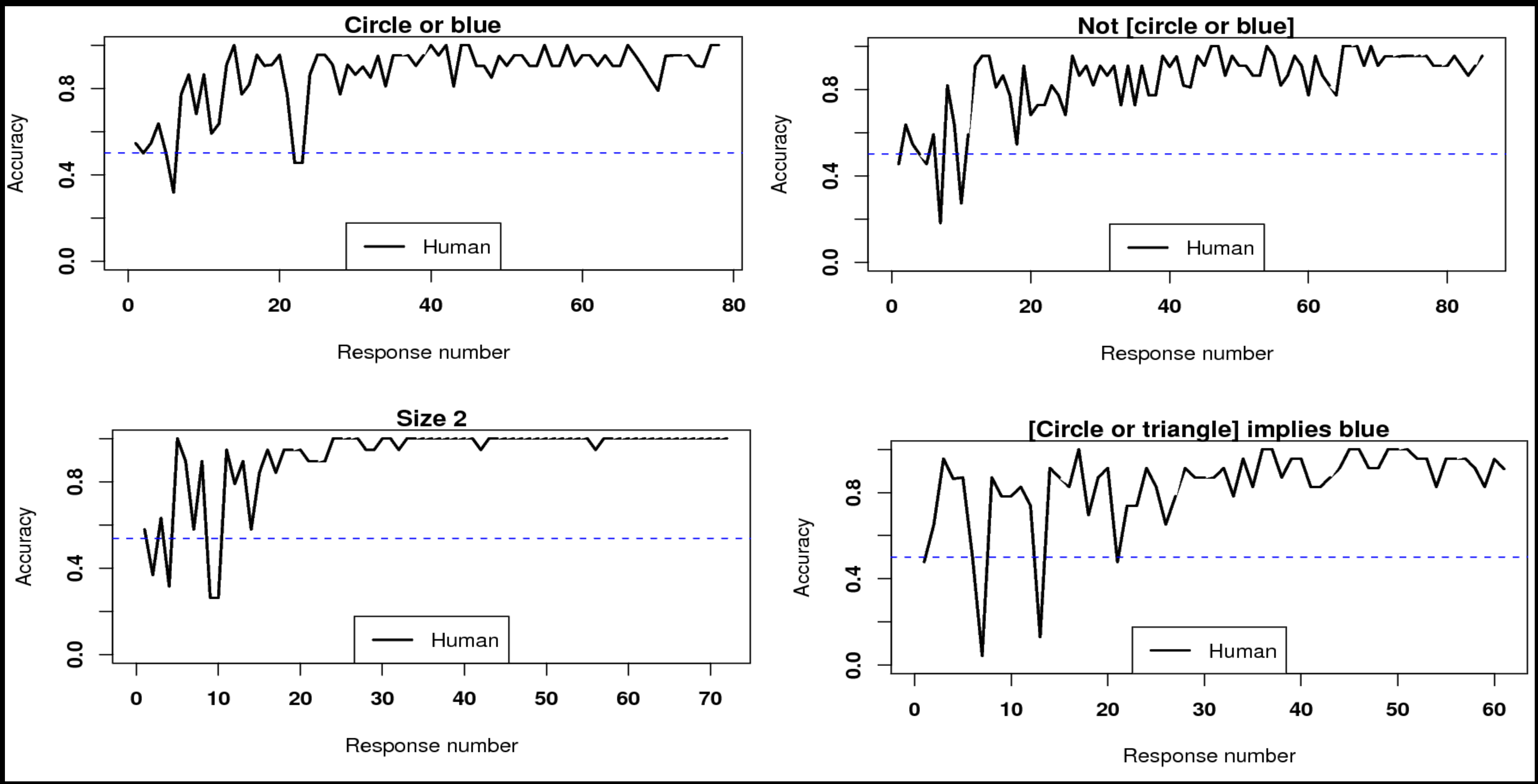
# Evaluating languages



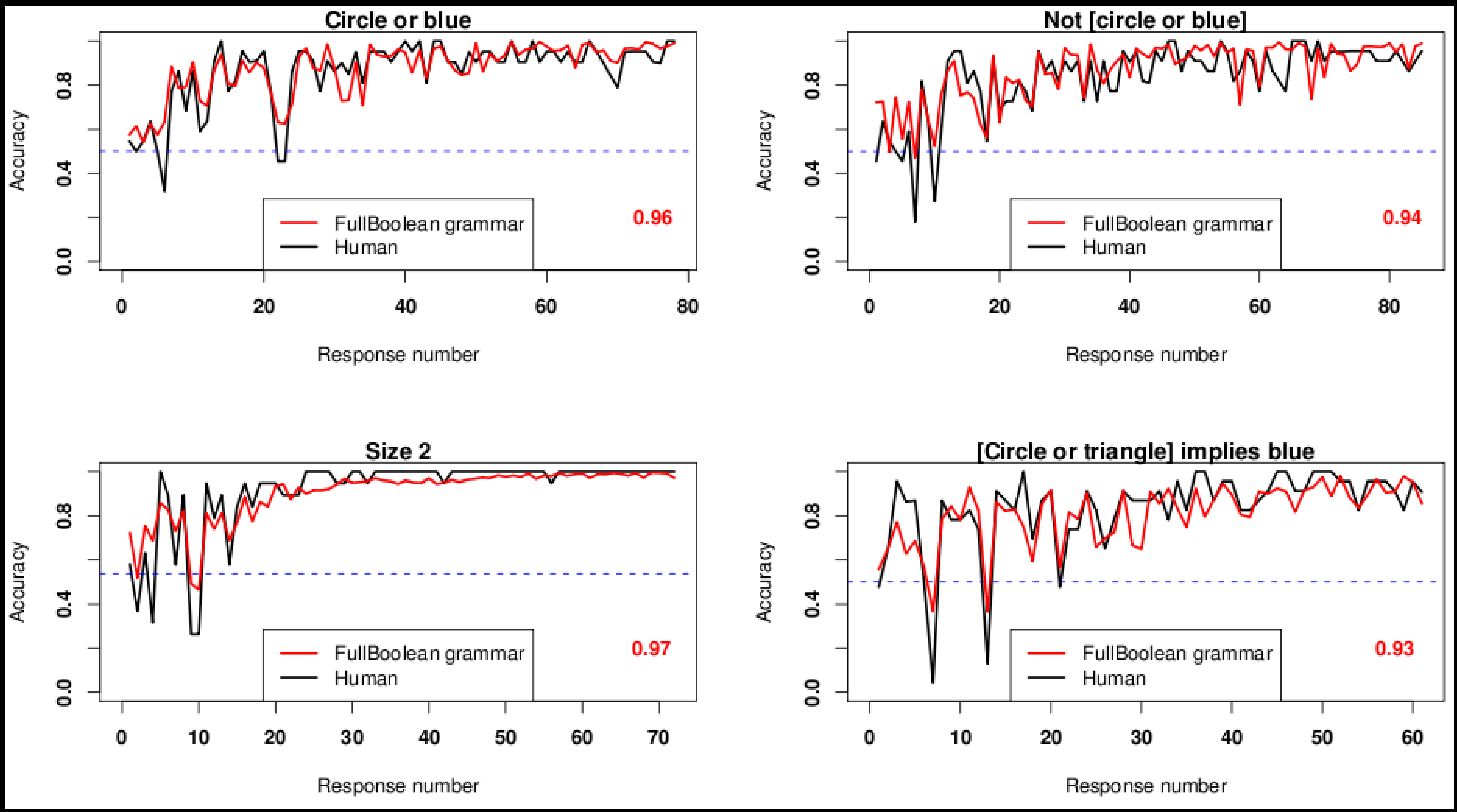

- High-throughput MTurk experiment.
  - 108 concepts,
    - Boolean (*circle or red*)
    - Context-dependent (“Determiners”)
      (*unique largest* , *exists another with same shape*)
  - 2 orders per concept,
  - 1596 participants.

Piantadosi, Goodman,
Tenenbaum (in prep)

# Boolean concepts

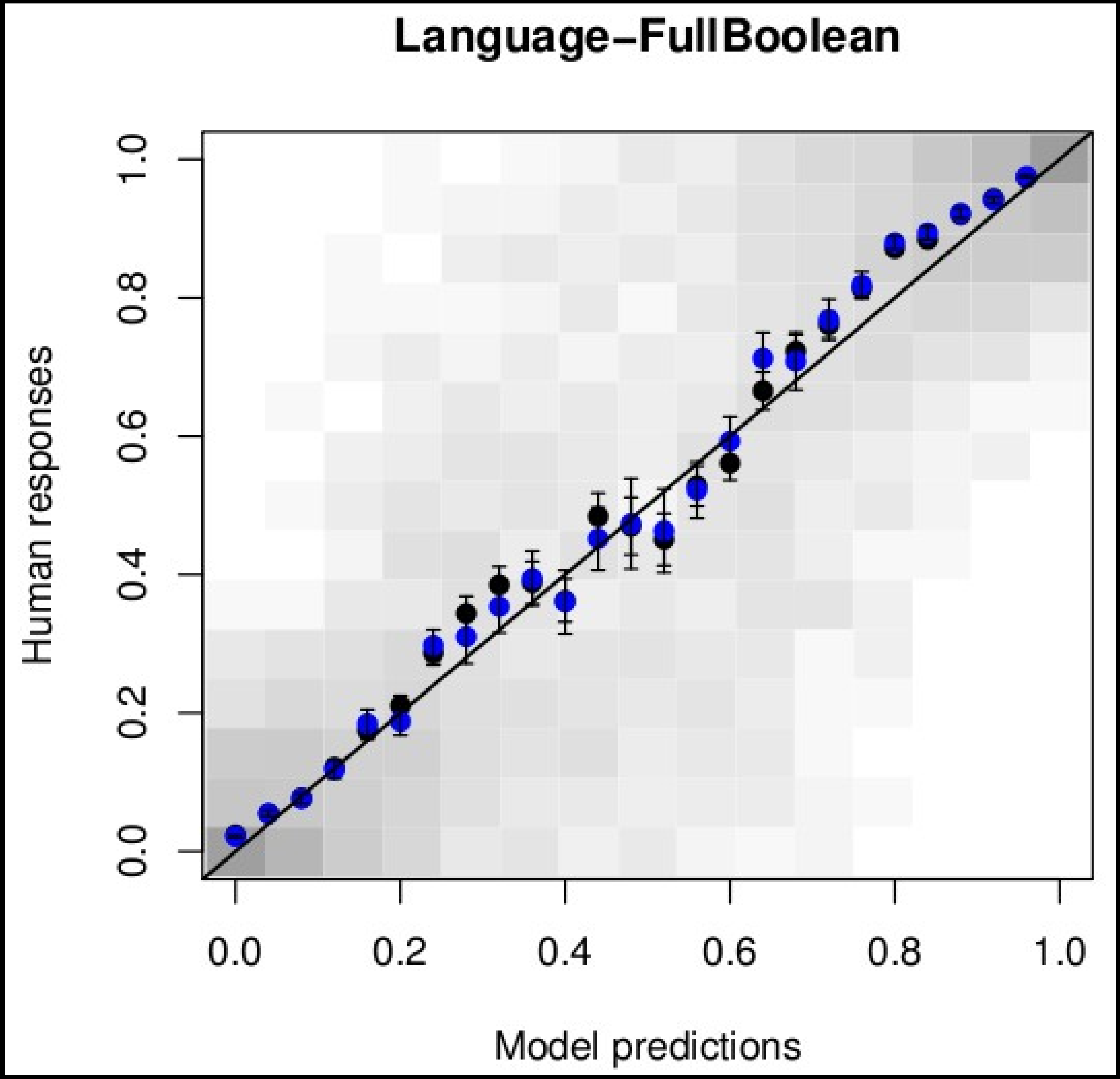


# Boolean concepts



# Boolean concepts

Best model performance on Boolean concepts:



# Comparing languages

- DNF
  disjunctions of conjunctions

```
(λ (x) (or (and (red? x) (circle? x))
(and (red? x) (triangle? x))))
```

- Horn clauses
  conjunctions of implications

```
(λ (x) (and (implies (not (red? x)) false)
(implies (not (triangle? x)) (circle? x))))
```

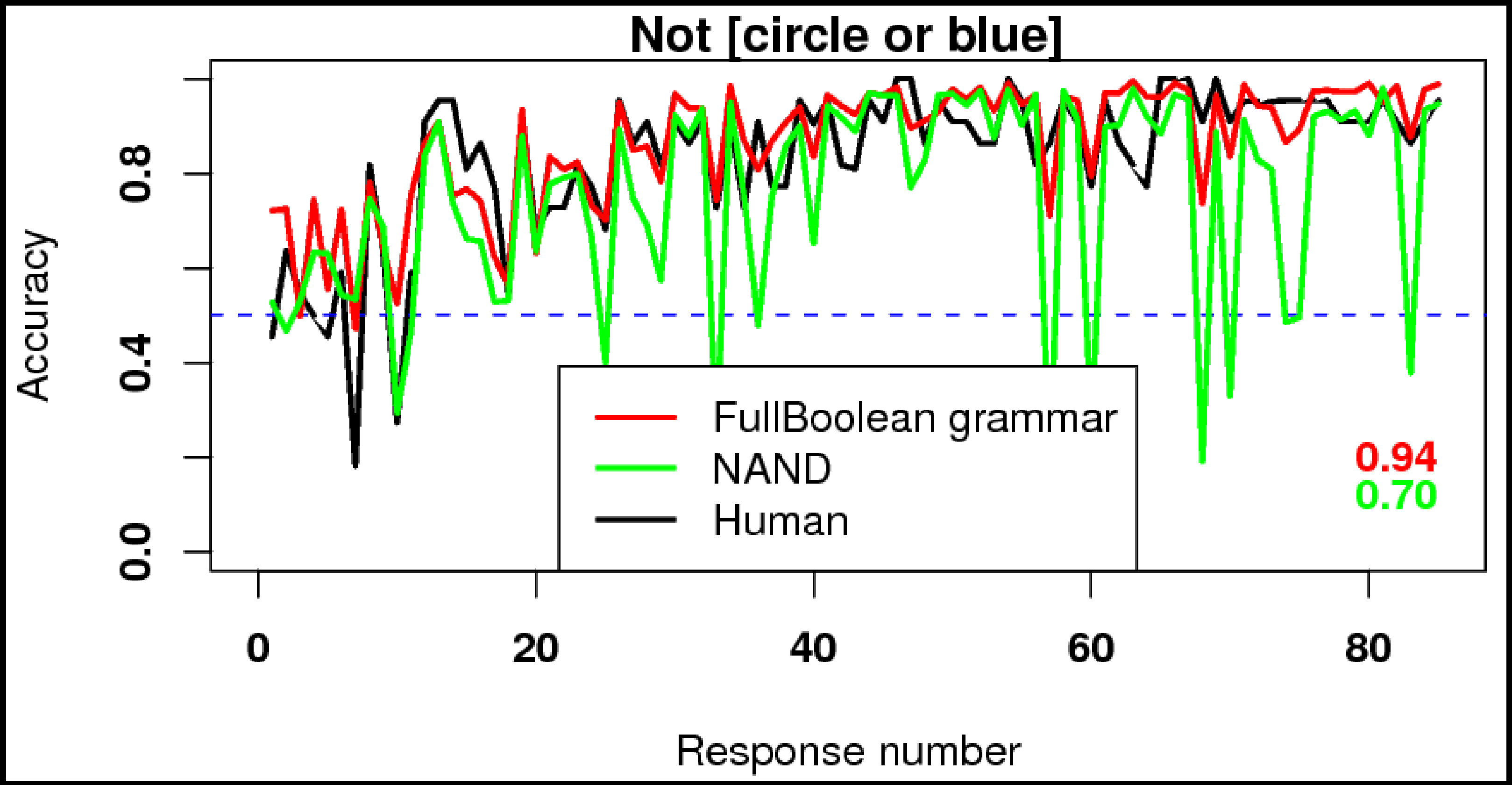
- Full boolean
  any combinations of AND, OR, NOT, IF, IFF

```
(λ (x) (and (red? x) (or (circle? x)
(triangle? x))))
```

- Nand
  combinations
  of NAND

```
(λ (x) (nand false (nand (red? x)
(nand (nand false (circle? x)) (nand
false (triangle? x))))))
```

# Comparing languages




- Fit hyper-parameters (dirichlet on each NT) for each language.
- Evaluated against held out data.

| Grammar | H.O. LL |
|---|---|
| FULLBOOLEAN | -16315.27 |
| CNF | -16333.59 |
| DNF | -16368.31 |
| BICONDITIONAL | -16385.01 |
| IMPLIES | -16442.40 |
| HORNCLAUSE | -16487.25 |
| SIMPLEBOOLEAN | -16490.51 |
| NAND | -16902.68 |
| NOR | -16917.49 |
| UNIFORM | -19482.72 |
| EXEMPLAR | -23645.13 |
| ONLYFEATURES | -31662.08 |
| RESPONSE-BIASED | -37906.77 |

# Outline

If concepts are probabilistic programs,
then concept learning is *probabilistic program induction*.

- Boolean categories
- Quantified concepts
- Natural number concepts
- Generative kinds
- Program induction

# Role-governed concepts

"Key"


Goodman, et al. (2007)

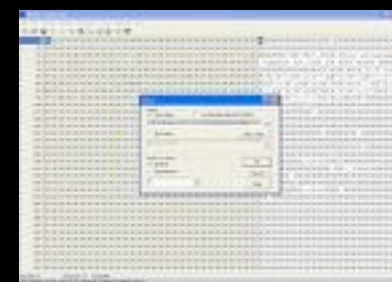
# Role-governed concepts

"Key"


Goodman, et al. (2007)

# Role-governed concepts

"Key"

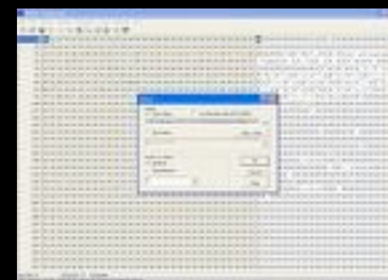



"There is no logic to the shape of a key. Its logic is: it turns the lock." -Chesterton

Goodman, et al. (2007)

# Role-governed concepts

"Key"


"There is no logic to the shape of a key. Its logic is: it turns the lock." -Chesterton

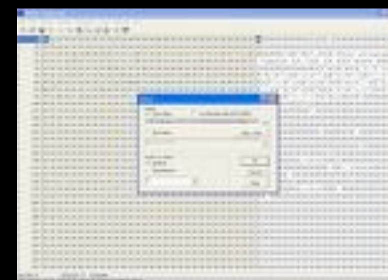
- Key, poison, passenger...

Goodman, et al. (2007)

# Role-governed concepts

“Key”





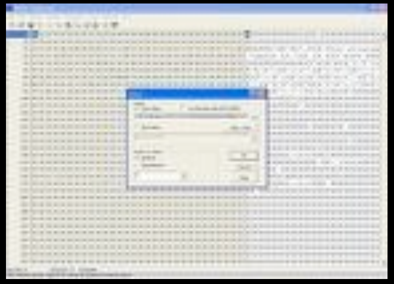
“There is no logic to the shape of a key. Its logic is: it turns the lock.” -Chesterton

- Key, poison, passenger...
  - Go beyond features to relations and *roles*.

Goodman, et al. (2007)

# Role-governed concepts

"Key"

"There is no logic to the shape of a key. Its logic is: it turns the lock." -Chesterton

- Key, poison, passenger...
  - Go beyond features to relations and *roles*.
  - Extend language by allowing relations and quantifiers.

Goodman, et al. (2007)

# Role-governed concepts

"Key"


"There is no logic to the shape of a key. Its logic is: it turns the lock." -Chesterton

- Key, poison, passenger...
  - Go beyond features to relations and *roles*.
  - Extend language by allowing relations and quantifiers.
- Features-to-relations shift (Cf. Keil & Batterman, 1984).

Goodman, et al. (2007)

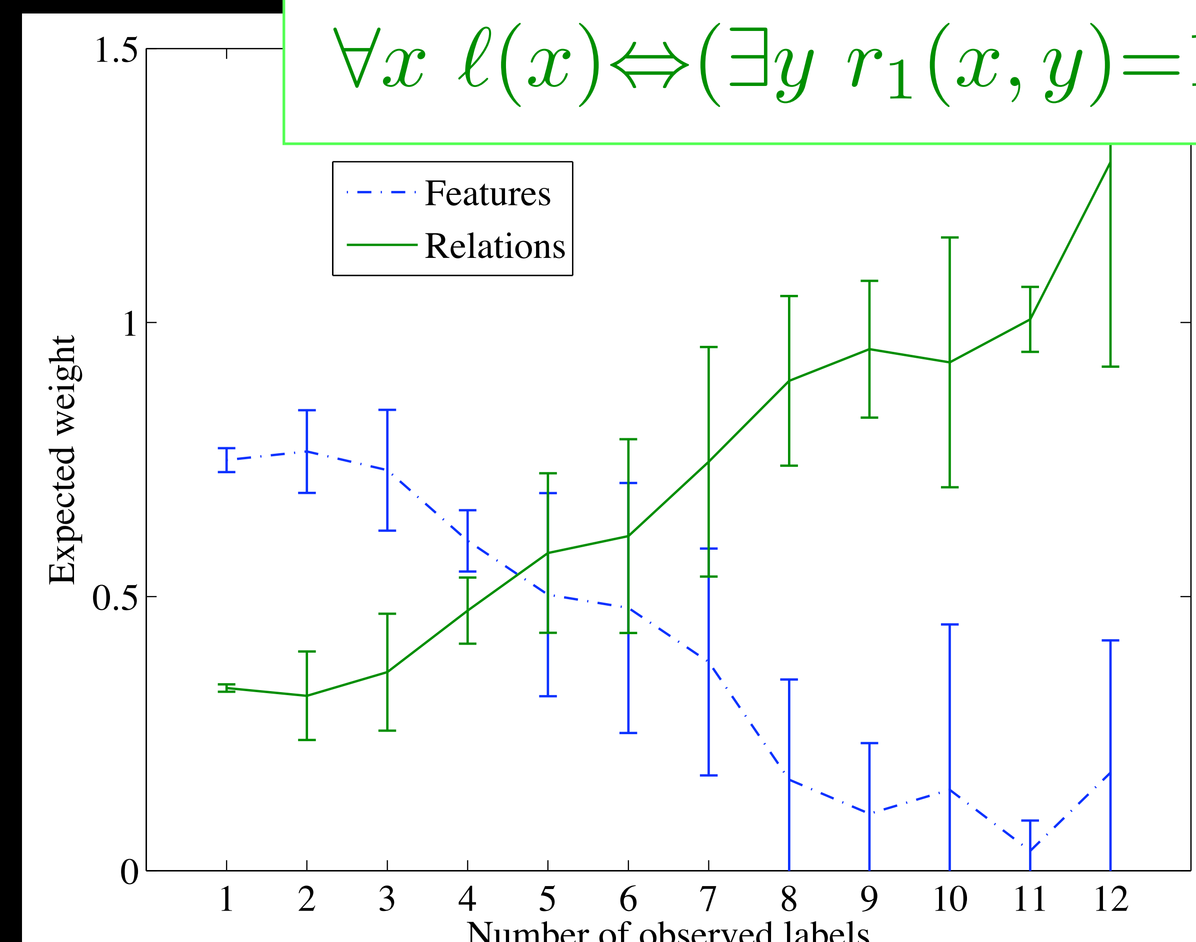
# Role-governed concepts

"Key"

"There is no logic to the shape of a key. Its logic is: it turns the lock." -Chesterton

- Key, poison, passenger...
  - Go beyond features to relations and *roles*.
  - Extend language by allowing relations and quantifiers.
- Features-to-relations shift (Cf. Keil & Batterman, 1984).

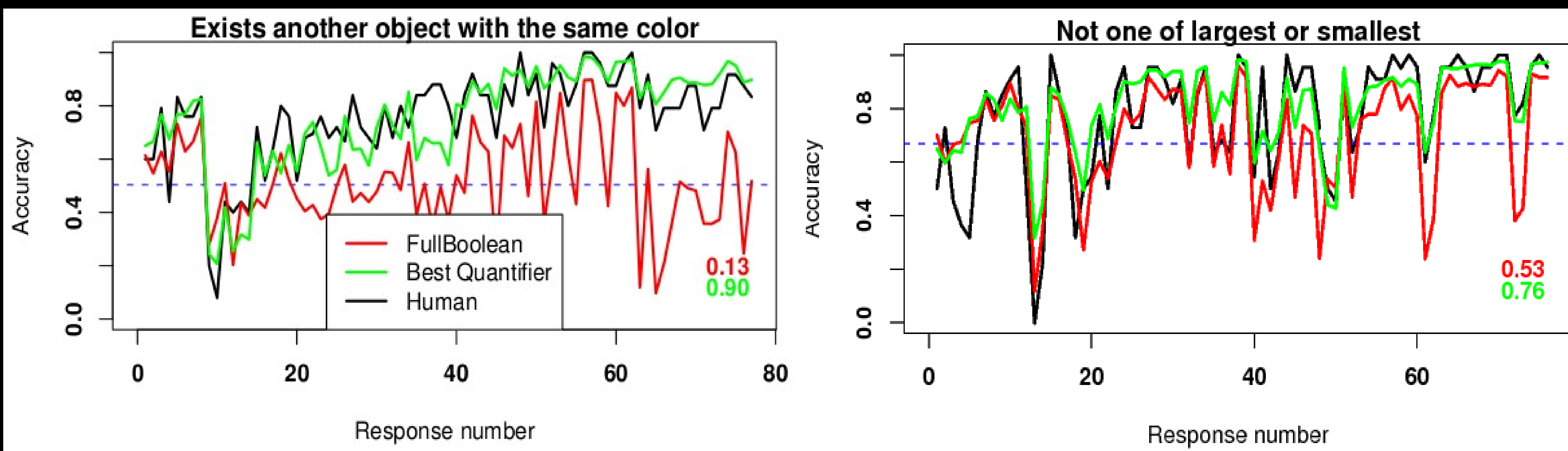
$\forall x\ \ell(x) \Leftrightarrow (\exists y\ r_1(x,y)=1)$




Goodman, et al. (2007)

# Non-Boolean concepts

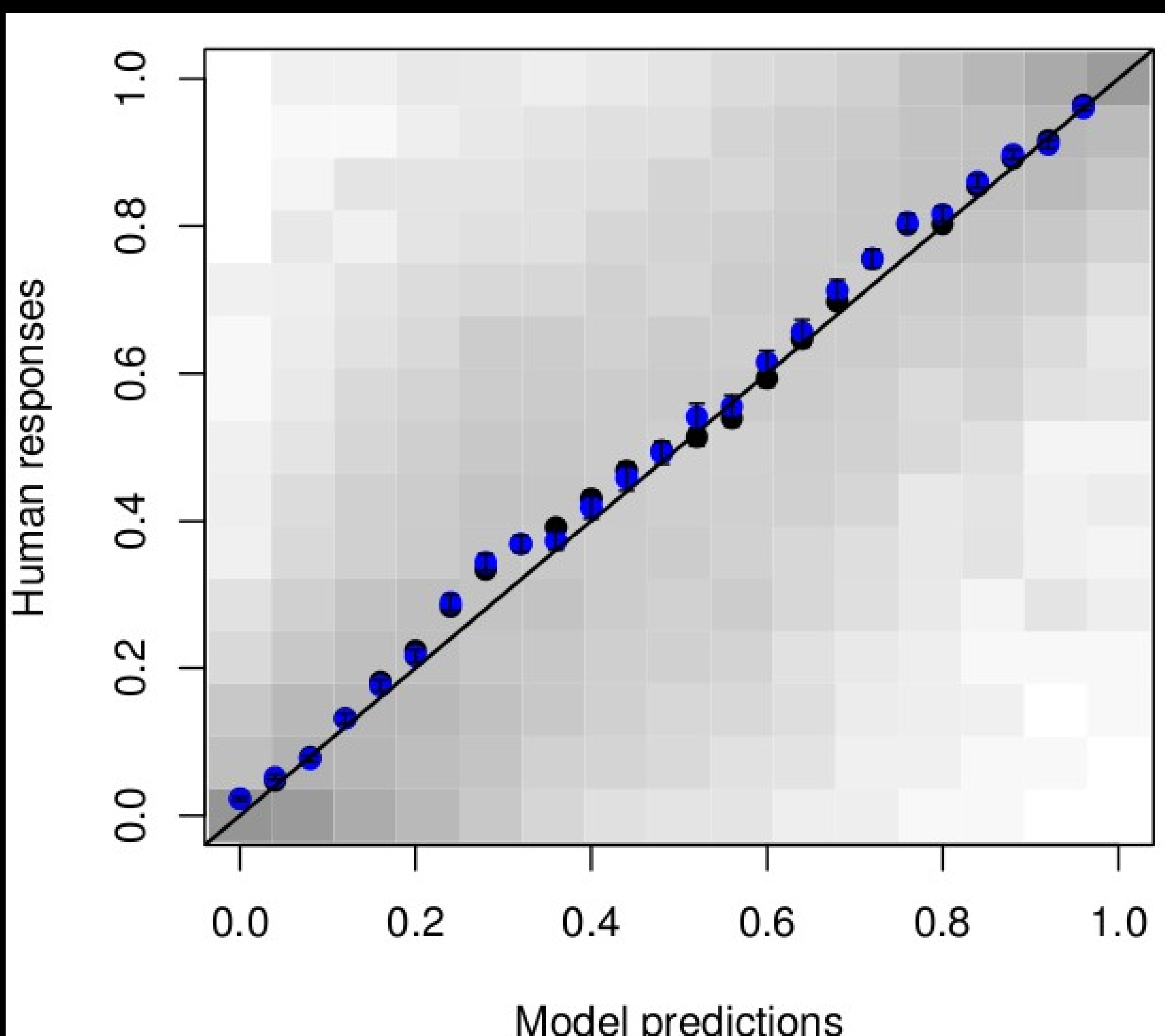
- Big experiment included context-dependent (determiner-like) concepts.
- What languages explain inductive bias for these non-boolean concepts?




Piantadosi, Goodman, Tenenbaum (in prep)

# Non-Boolean concepts

- Best language is full boolean **plus quantifiers**.




| FOL | One-Or-Fewer | Small-Cardinalities | 2nd-Ord.-Quan. | H.O. LL |
|---|---|---|---|---|
| ✓ | · | · | · | -79279.95 |
| ✓ | ✓ | · | · | -79560.90 |
| · | ✓ | · | · | -79642.46 |
| · | ✓ | ✓ | · | -79972.75 |
| ✓ | ✓ | · | ✓ | -80198.75 |
| ✓ | · | · | ✓ | -80267.46 |
| ✓ | · | ✓ | · | -80285.38 |
| · | ✓ | · | ✓ | -80300.00 |
| · | · | ✓ | · | -80614.35 |
| ✓ | ✓ | ✓ | · | -80942.77 |
| ✓ | ✓ | ✓ | ✓ | -81138.27 |
| · | ✓ | ✓ | ✓ | -81289.85 |
| ✓ | · | ✓ | ✓ | -81596.68 |
| · | · | ✓ | ✓ | -81651.36 |
| FULLBOOLEAN | | | | -81773.43 |
| BICONDITIONAL | | | | -81967.68 |
| SIMPLEBOOLEAN | | | | -82144.71 |
| · | · | · | · | -82219.08 |
| CNF | | | | -82685.21 |
| DNF | | | | -82752.82 |
| · | · | · | ✓ | -82853.59 |

# Other work

- Quantifying over objects/features
  (Kemp and Jerns, 2010)
- Learning a relation (by learning a theory)
  (Kemp, Goodman, Tenenbaum, 2008a, 2008b)
- Learning intuitive theories
  (Katz, et al, 2008; Goodman, Ullman, Tenenbaum, 2011; Ullman, Goodman, Tenenbaum, in prep)

# Outline

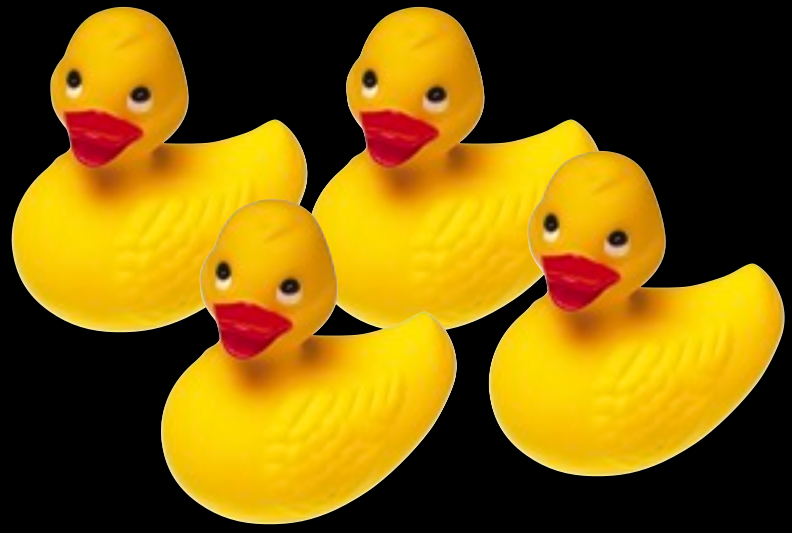
If concepts are probabilistic programs,
then concept learning is *probabilistic program induction*.

- Boolean categories
- Quantified concepts
- Natural number concepts
- Generative kinds
- Program induction

# Learning number

| Number knowledge | Age | Level |
|---|---|---|
| No word meanings | <24m | No-knower |

How many duckies?



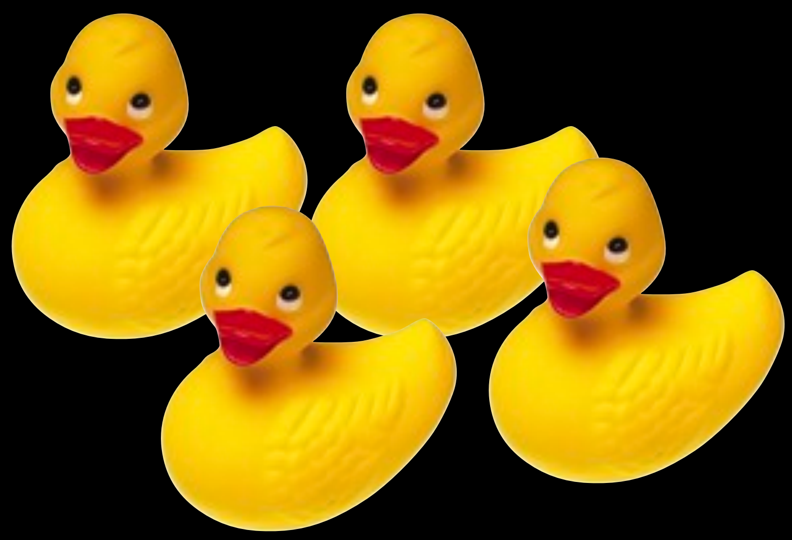
Can you give me two duckies?

See, e.g. Spelke 2003; Wynn 1990, 1992

# Learning number

| Number knowledge | Age | Level |
|---|---|---|
| No word meanings | <24m | No-knower |
| “one” | 24-30m | One-knower |

How many duckies?



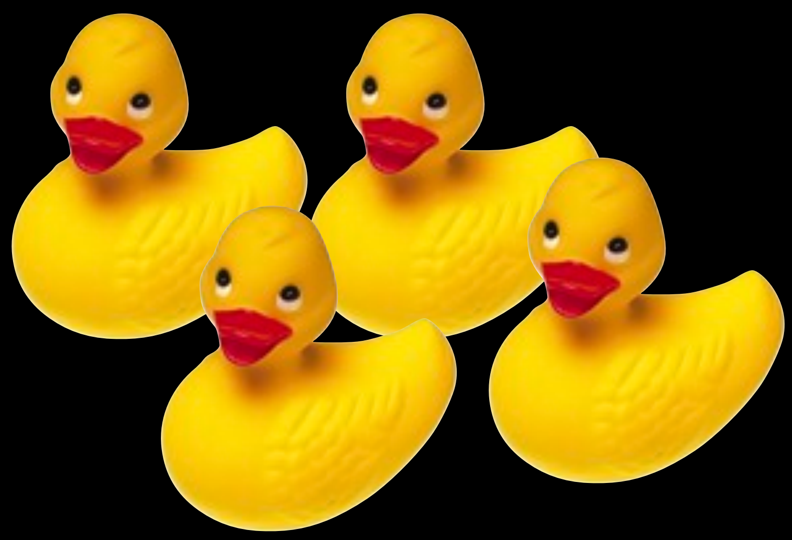
Can you give me two duckies?

See, e.g. Spelke 2003; Wynn 1990, 1992

# Learning number

| Number knowledge | Age | Level |
|---|---|---|
| No word meanings | <24m | No-knower |
| “one” | 24-30m | One-knower |
| “one”, “two” | 30-39m | Two-knower |

How many duckies?



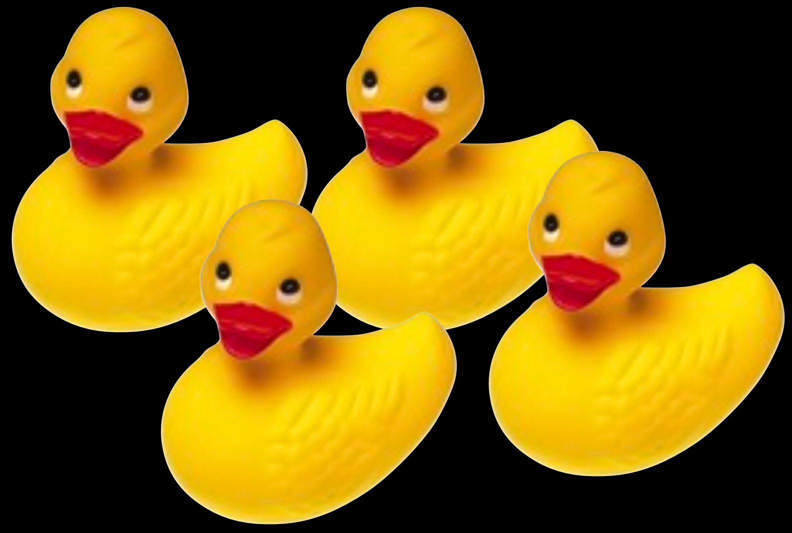
Can you give me two duckies?

See, e.g. Spelke 2003; Wynn 1990, 1992

# Learning number

| Number knowledge | Age | Level |
|---|---|---|
| No word meanings | <24m | No-knower |
| “one” | 24-30m | One-knower |
| “one”, “two” | 30-39m | Two-knower |
| “one”-”three” | 39-42m | Three-knower |

How many duckies?



Can you give me two duckies?

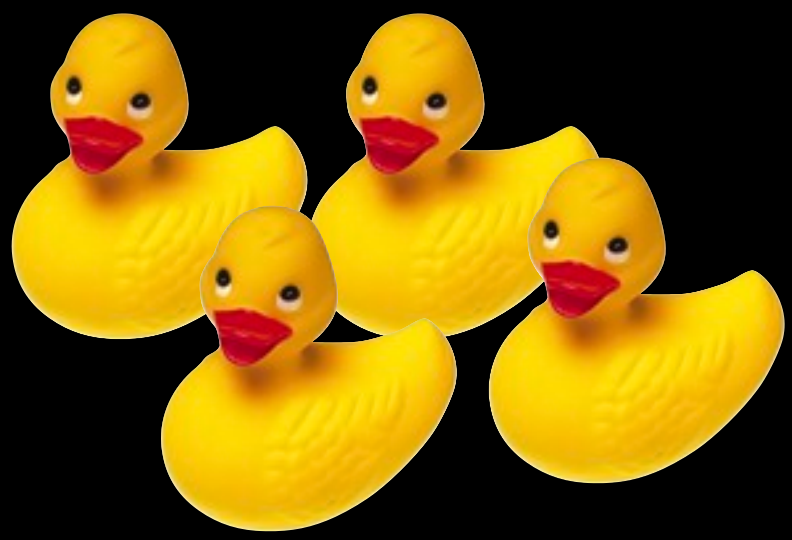
See, e.g. Spelke 2003; Wynn 1990, 1992

# Learning number

| Number knowledge | Age | Level |
| --- | --- | --- |
| No word meanings | <24m | No-knower |
| “one” | 24-30m | One-knower |
| “one”, “two” | 30-39m | Two-knower |
| “one”-”three” | 39-42m | Three-knower |
| All number words | >42m | CP-knower |

See, e.g. Spelke 2003; Wynn 1990, 1992

How many duckies?



Can you give me two duckies?

# Central questions

- How can number concepts be learned? (Cf. Rips, et al, 2008, and responses.)
  - In a way that doesn't presuppose integers?
  - Explaining the abrupt CP-transition?
  - What is the role of language?

# Learning number

- Sample a **lexicon**: a mapping from situations to descriptions.
- Lexicons expressed in (limited) λ-calculus plus primitives:
  - Set primitives: `difference`, `union`, `select`, `singleton?`, `doubleton?`, ...
  - Count-list operations: `prev` / `next` move between *words* on the list.
  - Recursion: `(L S)`.
  - `if`, `and`, ...

Piantadosi, Tenenbaum, Goodman (subm.)

# Learning number

A two-knower lexicon:

```
(define L
 (λ (S)
   (if (singleton? S)
       “one”
       (if (doubleton? S) “two” undef))
```

# Learning number

A two-knower lexicon:

```
(define L
 (λ (S)
   (if (singleton? S)
       “one”
       (if (doubleton? S) “two” undef))
```

A CP-knower lexicon:

```
(define L
 (λ (S)
   (if (singleton? S)
       “one”
       (next (L (set-difference S (select S)))))
```

# Learning number

- Large space of hypotheses contains many potentially useful lexica, as well as very silly ones.

# Learning number

- Large space of hypotheses contains many potentially useful lexica, as well as very silly ones.

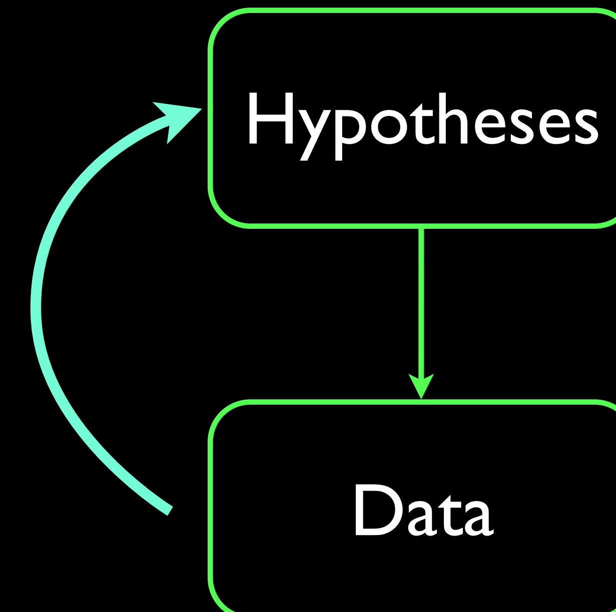
For example: a ‘mod 5’ lexicon:

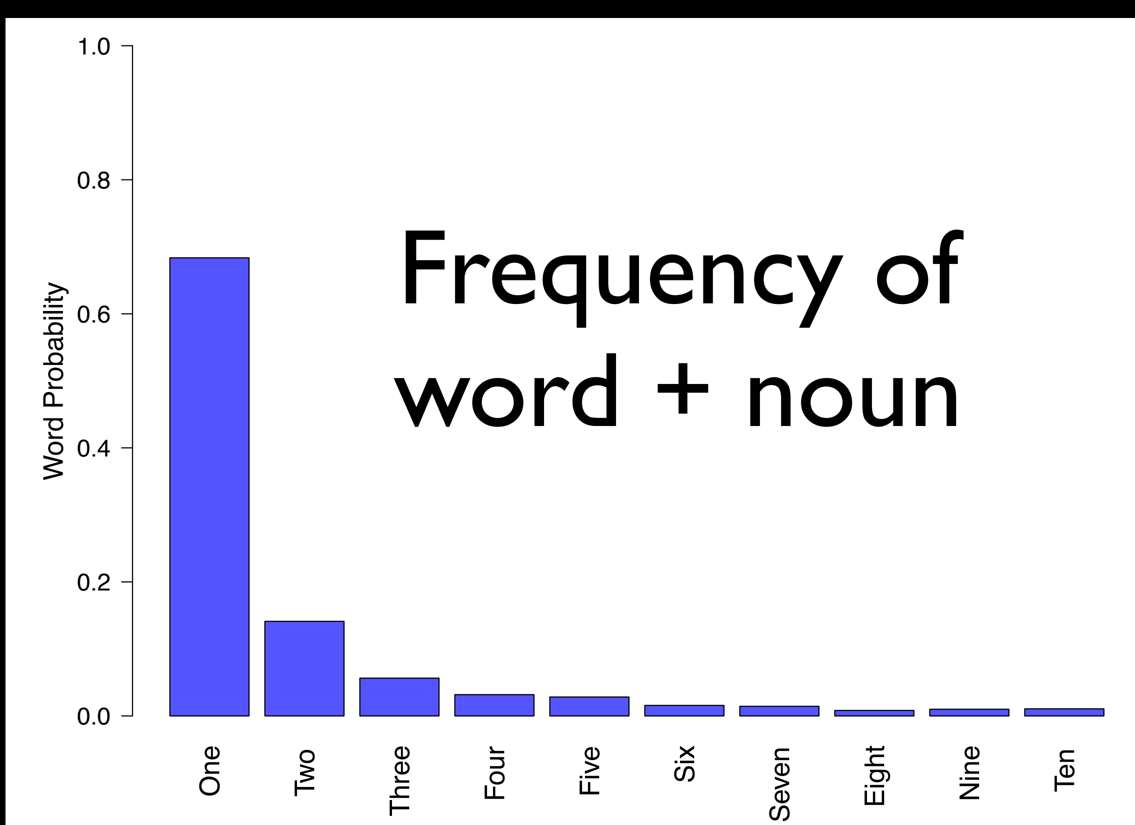
```
(define L
 (λ (S)
   (if (or (singelton? S)
           (equal? (L (set-diff S (select S)))
                   “five”)
        “one”
        (next (L (set-diff S (select S)))))))
```

(Cf. Rips, et al, 2008.)

# Learning number

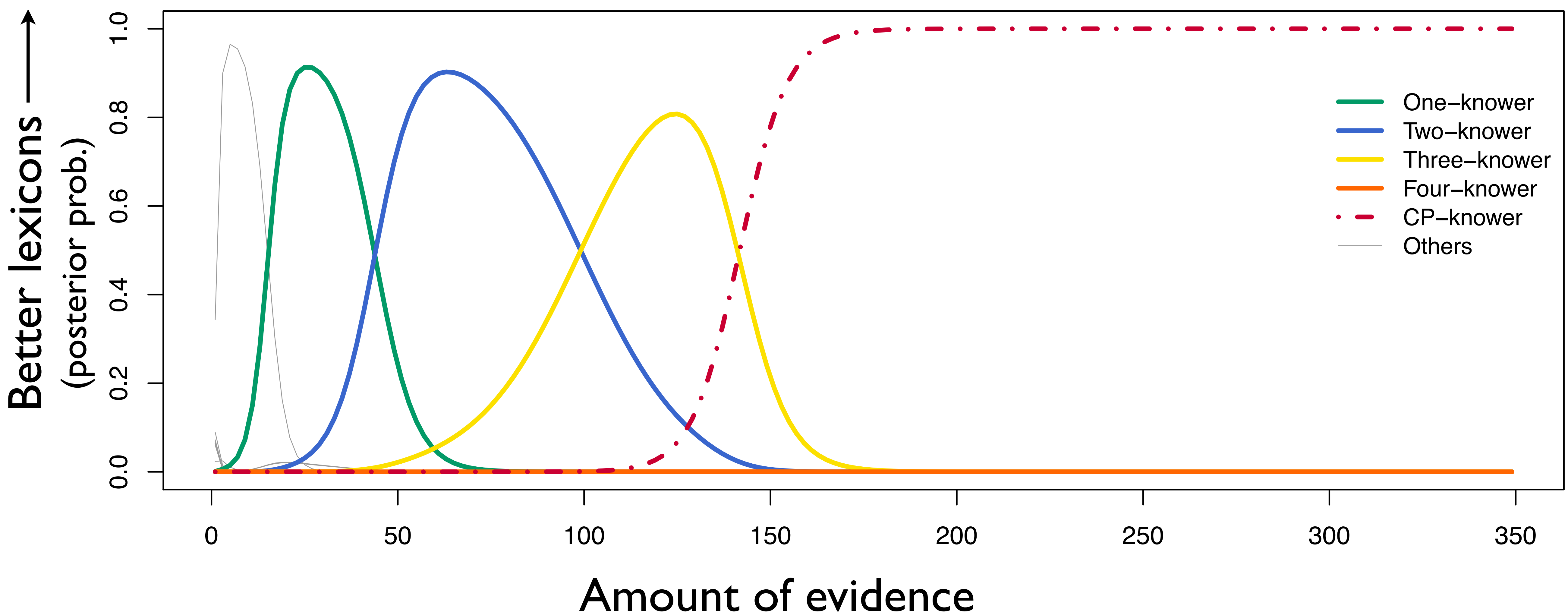
```
(query
 (define lexicon (noisify (lex-generator))
 (eq? (lexicon        )"two blobs!" )
 (and (eq? (lexicon      ) "one blobs!")
      (eq? (lexicon      ) "two blobs!")
      ...))
```



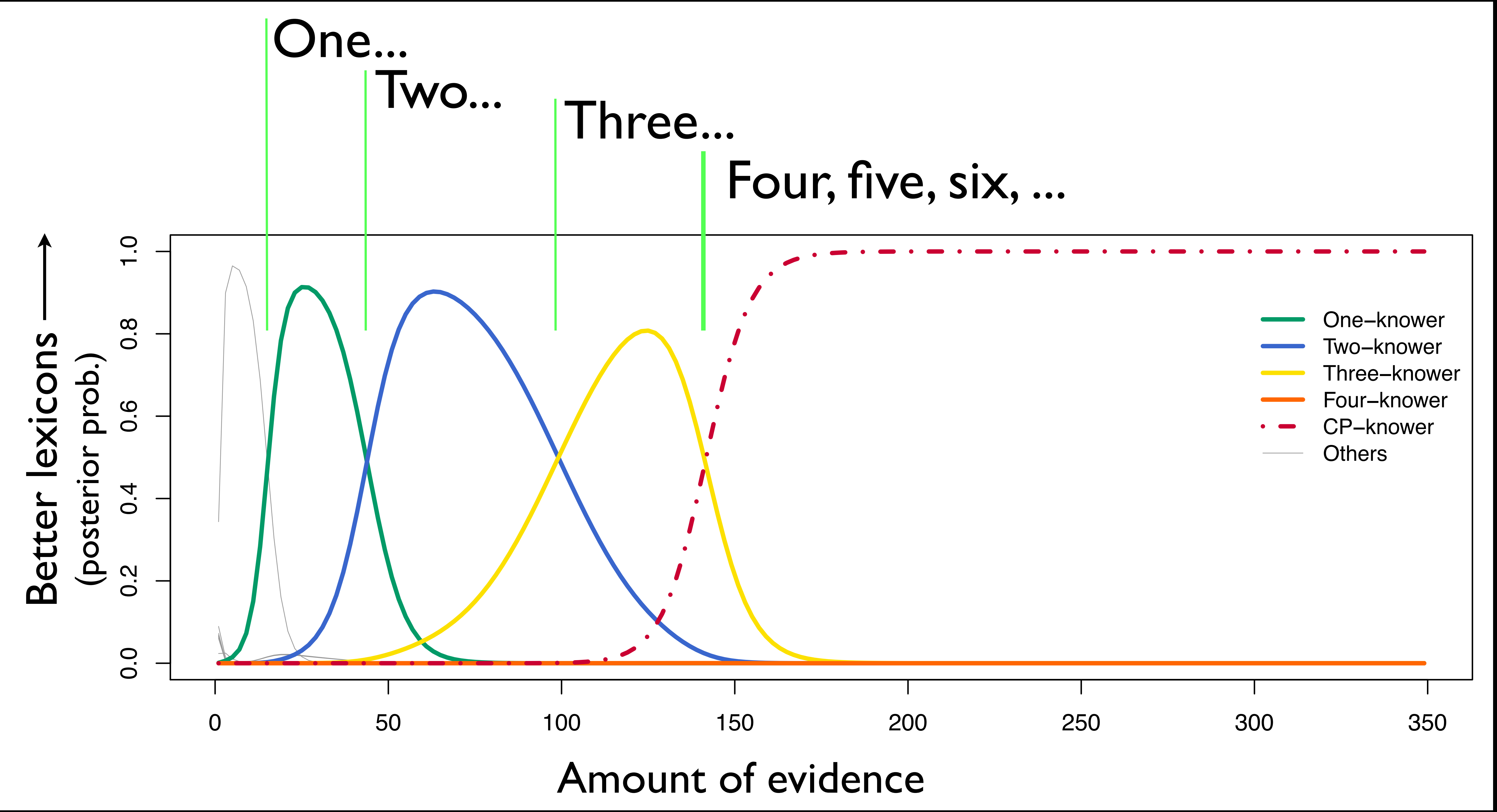




- Learning data: number words paired with sets of objects (frequency of words matches CHILDES corpus).

# Learning number



# Learning number

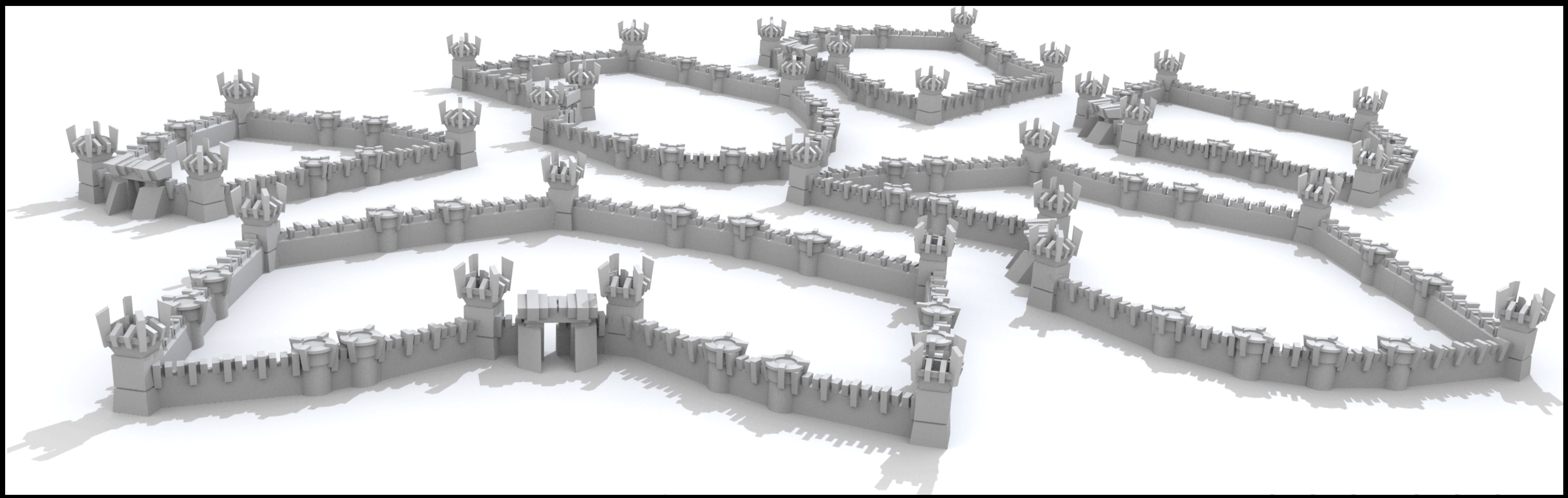


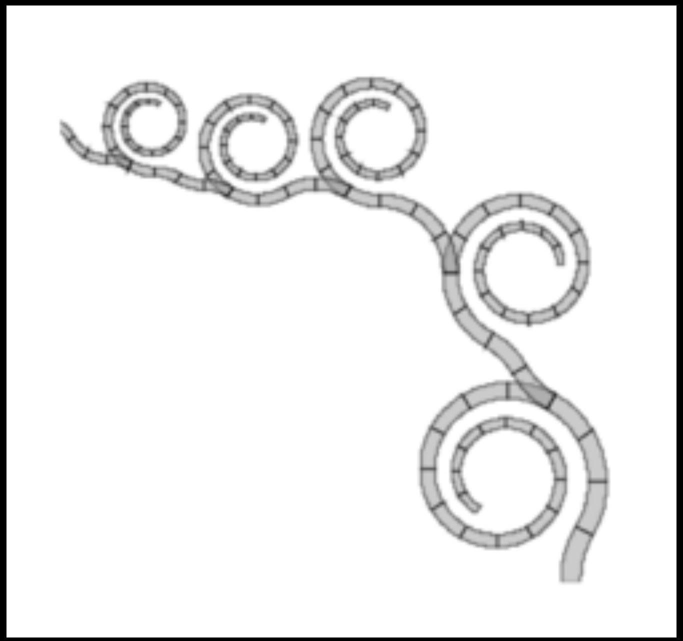
# Outline

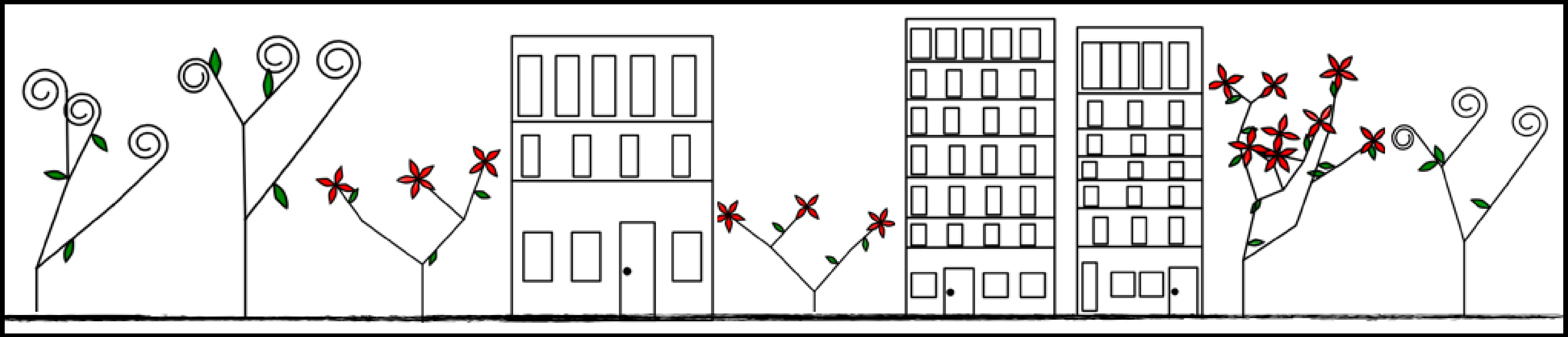
If concepts are probabilistic programs,
then concept learning is *probabilistic program induction*.

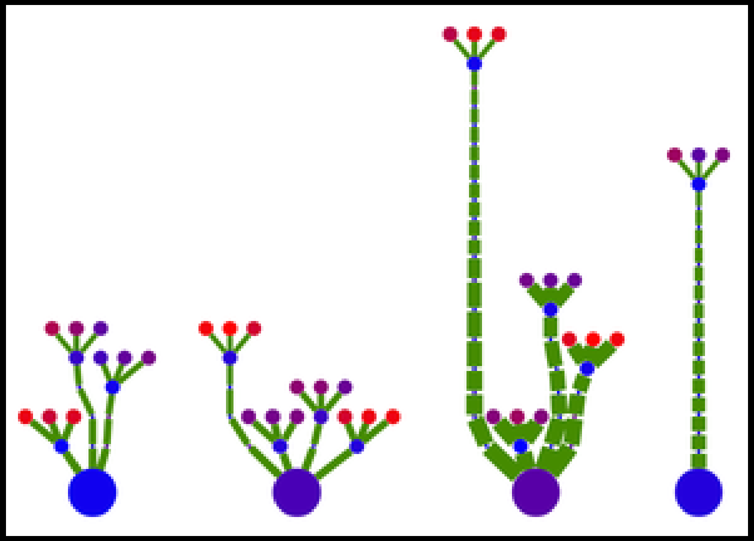
- Boolean categories
- Quantified concepts
- Natural number concepts
- Generative kinds
- Program induction

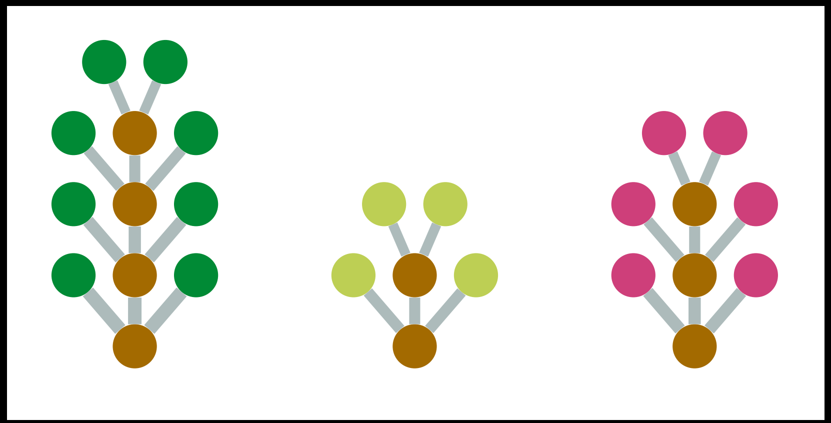
# Generative kinds







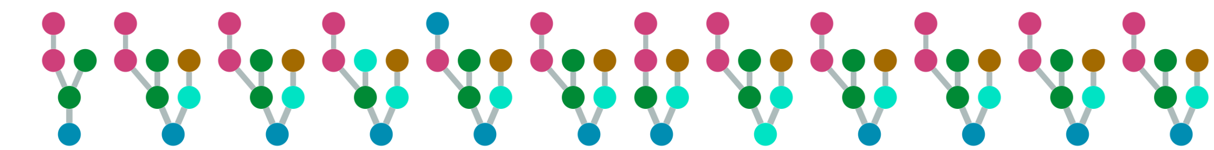


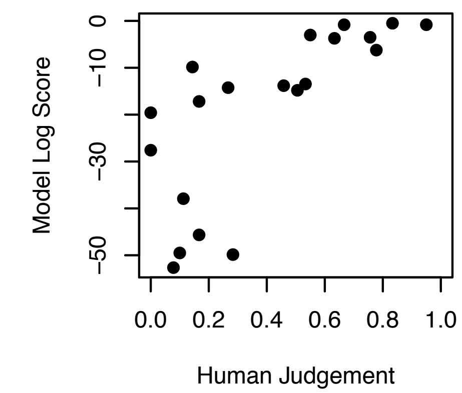


# Learning generative kinds

## Simple prototype

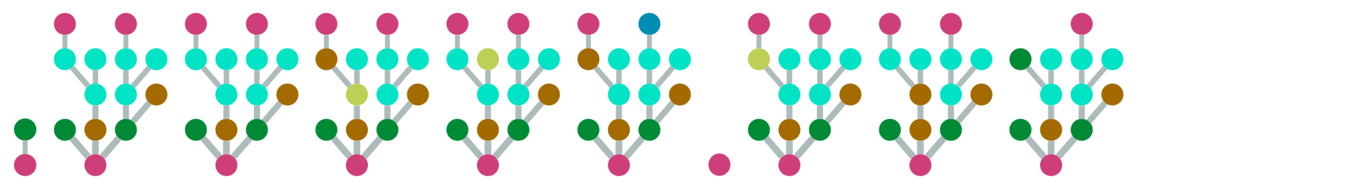
```
(lambda ()
  (node 'a
        (node 'b (node 'c (node 'c)) (node 'b))
        (node 'd (node 'e))))
```

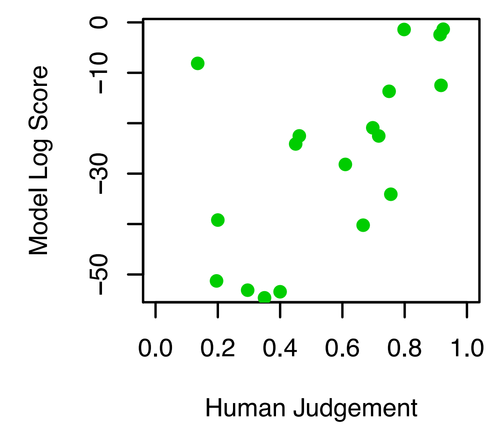





## Sub-concepts

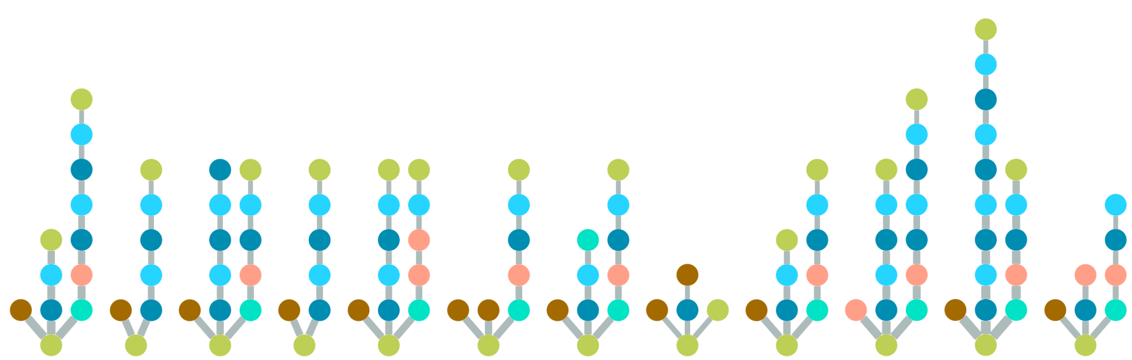
```
(begin
  (define (part) (node 'c (node 'c (node 'a)) (node 'c)))
  (lambda () (node 'a (node 'b) (node 'd (part)) (node 'b (part) (node 'd)))))
```

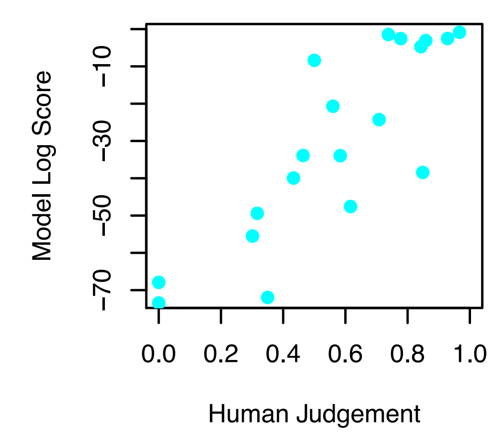





## Single recursion

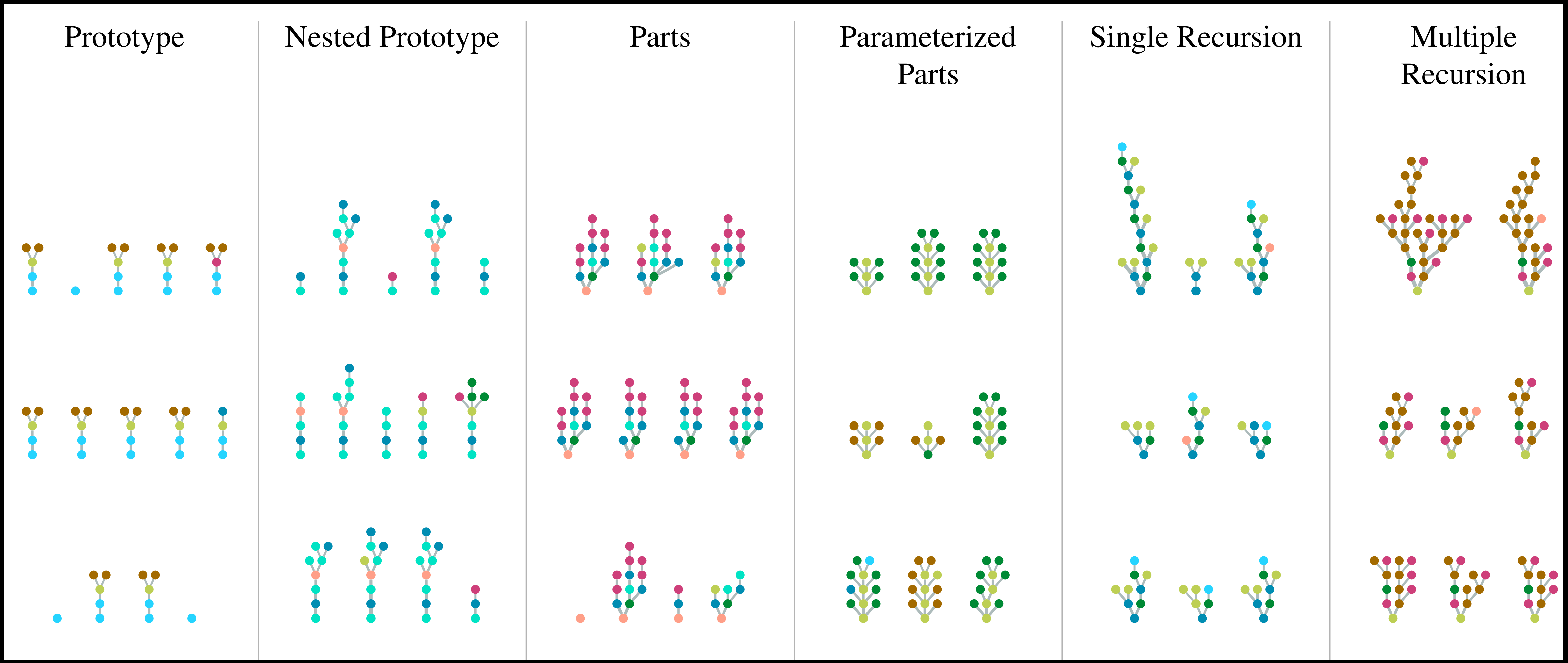
```
(begin
  (define (part) (node 'a (node 'b (if (flip) (node 'c) (part)))))
  (lambda () (node 'c (node 'd) (part) (node 'e (node 'f (part))))))
```






Stuhlmueller, Tenenbaum, Goodman (2010)

# Learning generative kinds




| | Set GCM | Transition GCM | Tree GCM | Generative Model |
|---|---|---|---|---|
| Prototype | 0.589 | 0.751 | **0.803** | 0.748 |
| Nested Prototype | 0.544 | 0.851 | **0.937** | 0.904 |
| Parts* | 0.320 | 0.617 | 0.705 | **0.835** |
| Parameterized Parts | 0.298 | 0.591 | 0.778 | **0.911** |
| Single Recursion | 0.284 | 0.499 | 0.637 | **0.773** |
| Multiple Recursion | 0.505 | 0.561 | 0.451 | **0.770** |

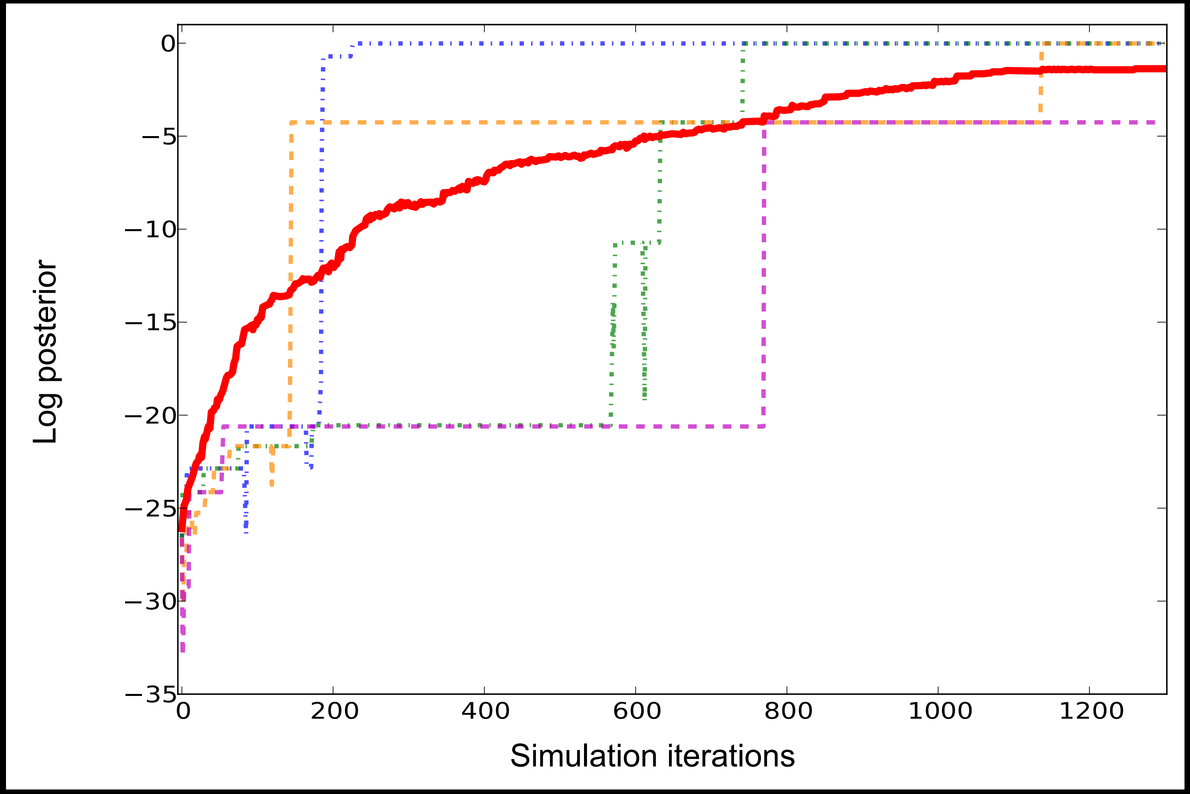
# Outline

If concepts are probabilistic programs,
then concept learning is *probabilistic program induction*.

- Boolean categories
- Quantified concepts
- Natural number concepts
- Generative kinds
- Program induction

# Algorithms for induction

- What algorithms are capable of learning concepts in a language of thought?
  - All results so far were computed using MCMC based on constituent regeneration (Goodman, et al, 2008).
  - Is this cognitively plausible? Maybe...
  - But this is probably not enough on its own.




(Ullman, Goodman, Tenenbaum, 2010)

# Program induction

- Program induction is especially hard. How could it be done?
- Idea: syntactic analogy + argument compression (+ search/MCMC).

```
(/ (+ 1 (* 2 3))
   (+ 1 (* 5 3)))
```

"inverse inlining" →

```
(define (F x) (+ 1 (* x 3)))
(/ (F 2) (F 5))
```

```
(define (F x)
 (+ 1 (* x 3)))
(/ (F 2) (F 5))
```

"de-argument" →

```
(define (F)
 (+ 1 (* (gaussian 3.5 1.0) 3)))
(/ (F) (F))
```

```
(define (F x)
 (+ 1 (* x 3)))
(F (F 1))
```

"de-argument" →

```
(define (F)
 (if (flip) (+ 1 (* (F) 3)) 1))
(F)
```

# Program induction

- Program induction is especially hard. How could it be done?
- Idea: syntactic analogy + argument compression (+ search/MCMC).

```
(/ (+ 1 (* 2 3))
   (+ 1 (* 5 3)))
```

"inverse inlining" →

```
(define (F x) (+ 1 (* x 3)))
(/ (F 2) (F 5))
```

```
(define (F x)
 (+ 1 (* x 3)))
(/ (F 2) (F 5))
```

"de-argument" →

```
(define (F)
 (+ 1 (* (gaussian 3.5 1.0) 3)))
(/ (F) (F))
```

```
(define (F x)
 (+ 1 (* x 3)))
(F (F 1))
```

"de-argument" →

```
(define (F)
 (if (flip) (+ 1 (* (F) 3)) 1))
(F)
```

# Program induction

- Program induction is especially hard. How could it be done?
- Idea: syntactic analogy + argument compression (+ search/MCMC).

```
(/ (+ 1 (* 2 3))
   (+ 1 (* 5 3)))
```

"inverse inlining" →

```
(define (F x) (+ 1 (* x 3)))
(/ (F 2) (F 5))
```

```
(define (F x)
 (+ 1 (* x 3)))
(/ (F 2) (F 5))
```

"de-argument" →

```
(define (F)
 (+ 1 (* (gaussian 3.5 1.0) 3)))
(/ (F) (F))
```

```
(define (F x)
 (+ 1 (* x 3)))
(F (F 1))
```

"de-argument" →

```
(define (F)
 (if (flip) (+ 1 (* (F) 3)) 1))
(F)
```

# Program induction

- Program induction is especially hard. How could it be done?
- Idea: syntactic analogy + argument compression (+ search/MCMC).

```
(/ (+ 1 (* 2 3))
   (+ 1 (* 5 3)))
```

“inverse inlining” →

```
(define (F x) (+ 1 (* x 3)))
(/ (F 2) (F 5))
```

```
(define (F x)
 (+ 1 (* x 3)))
(/ (F 2) (F 5))
```

“de-argument” →

```
(define (F)
 (+ 1 (* (gaussian 3.5 1.0) 3)))
(/ (F) (F))
```

```
(define (F x)
 (+ 1 (* x 3)))
(F (F 1))
```

“de-argument” →

```
(define (F)
 (if (flip) (+ 1 (* (F) 3)) 1))
(F)
```

# Program induction

- Program induction is especially hard. How could it be done?
- Idea: syntactic analogy + argument compression (+ search/MCMC).

```
(/ (+ 1 (* 2 3))
   (+ 1 (* 5 3)))
```

"inverse inlining" →

```
(define (F x) (+ 1 (* x 3)))
(/ (F 2) (F 5))
```

```
(define (F x)
 (+ 1 (* x 3)))
(/ (F 2) (F 5))
```

"de-argument" →

```
(define (F)
 (+ 1 (* (gaussian 3.5 1.0) 3)))
(/ (F) (F))
```

```
(define (F x)
 (+ 1 (* x 3)))
(F (F 1))
```

"de-argument" →

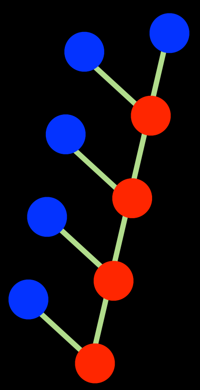
```
(define (F)
 (if (flip) (+ 1 (* (F) 3)) 1))
(F)
```

# Program induction

- *Bayesian program merging* algorithm:
  (Cf. Stolcke & Omohundro, 1994)
  - Initial state is exemplar program (mixture of data).
  - Transform programs via syntactic analogy and argument compression.
  - Beam search* with respect to the posterior score.
    (*Or stochastic search, Monte Carlo, etc.)
    - Likelihood: marginal probability of data given program (computed by lazy particle filter).
    - Prior: syntactic complexity.

Hwang, Stuhlmueller, Goodman (in prep)

# Example



"data incorporation"

```
(node r (node b)
        (node r (node b)
                (node r (node b)
                        (node r (node b) b)
```

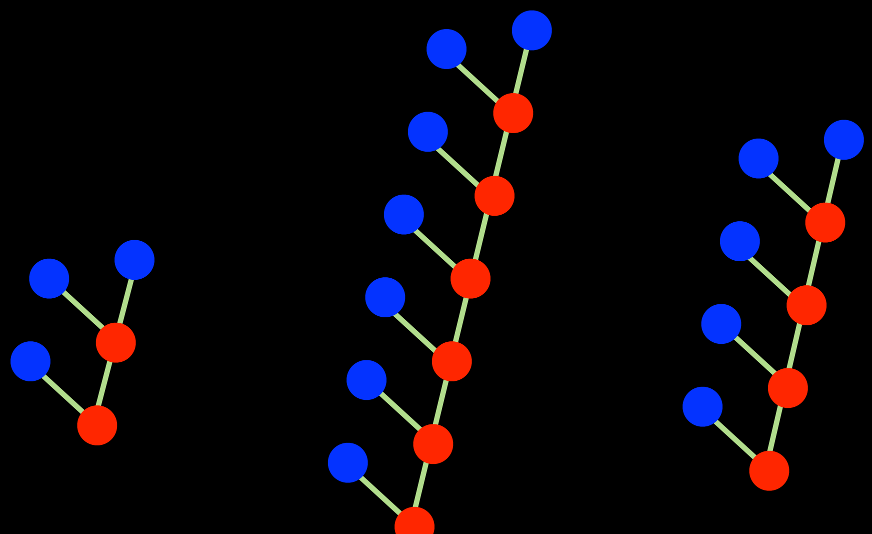
"inverse inline"

```
(define (F x) (node r (node b) x))
(F (F (F (F b))))
```

"de-argument"

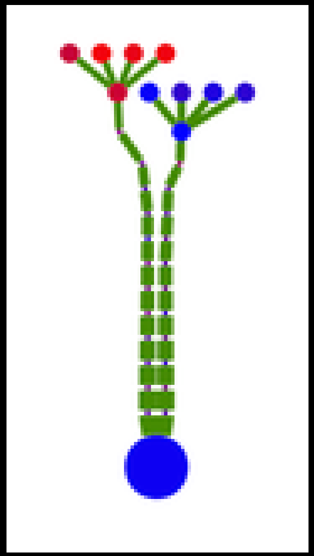
```
(define (F) (if (flip 0.8)
                (node r (node b) (F))
                b))
(F)
```

sample

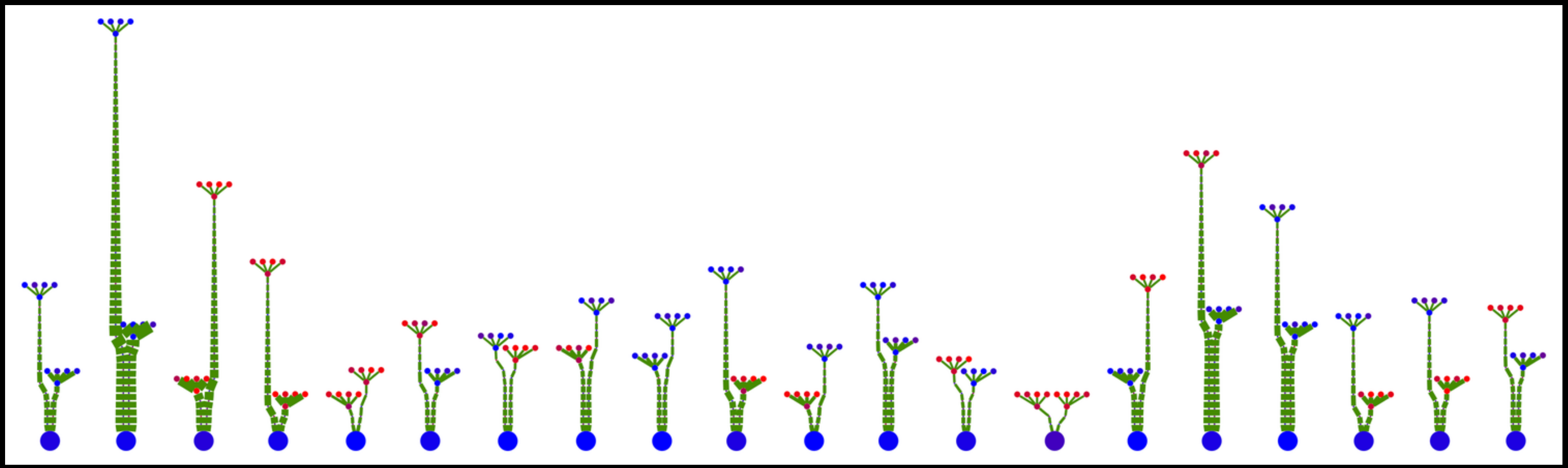


...

# Example



```
(begin (define F4 (lambda (V7 V8) (node (F1 V7 0.1) V8)))
    (define F3 (lambda (V6) (node (F1 V6 0.3))))
    (define F2
      (lambda (V3 V4)
        ((lambda (V5) (F4 V3 (F4 V4 V5)))
          (if (flip 9/11)
              (F2 81.0 85.0)
              (uniform-choice
                     (node (F1 204.0 0.3) (F3 199.0)
                       (F3 243.0) (F3 233.0) (F3 240.0)))
                     (F4 151.0
                       (node (F1 -21.0 0.3) (F3 7.0)
                         (F3 49.0) (F3 3e1) (F3 44.0))))))))
    (define F1
      (lambda (V1 V2)
        (data (color (gaussian V1 25)) (size V2))))
    (lambda ()
      (uniform-choice
             (node (F1 13.0 1) (F2 89.0 111.0)
               (F2 85.0 121.0)))))
```



# Analogy and number?

- In simple MCMC over a rich concept space some transitions are very unlikely.
  - E.g. N-knower => CP-knower.

# Analogy and number?

- In simple MCMC over a rich concept space some transitions are very unlikely.
  - E.g. N-knower => CP-knower.
- Proposals via Syntactic analogy + recursion detection?

# Analogy and number?

- In simple MCMC over a rich concept space some transitions are very unlikely.
  - E.g. N-knower => CP-knower.
- Proposals via Syntactic analogy + recursion detection?

A three-knower lexicon:

```
(define lexicon
 (λ (word)
  (case word
   (“one” (λ (set)
          (empty? (diff set (select set))))
   (“two”   (λ (set)
             ((lexicon “one”)
              (diff set (select set))
   (“three” (λ (set)
             ((lexicon “two”)
              (diff set (select set))
   (else null))
```

# Analogy and number?

- In simple MCMC over a rich concept space some transitions are very unlikely.
  - E.g. N-knower => CP-knower.
- Proposals via Syntactic analogy + recursion detection?

A three-knower lexicon:

```
(define lexicon
 (λ (word)
  (case word
   (“one” (λ (set)
            (empty? (diff set (select set))))
   (“two”   (λ (set)
             ((lexicon “one”)
              (diff set (select set)))
   (“three” (λ (set)
             ((lexicon “two”)
              (diff set (select set)))
   (else null))
```

# Analogy and number?

- In simple MCMC over a rich concept space some transitions are very unlikely.
  - E.g. N-knower => CP-knower.
- Proposals via Syntactic analogy + recursion detection?

A CP-knower lexicon:

```
(define lexicon
 (λ (word)
  (if (= word “one”)
      (λ (set)
       (empty? (diff set (select set))))
      (λ (set)
        ((lexicon (prev word))
                 (diff set (select set)))))))
```

```
(
   (“three” (λ (set)
             ((lexicon “two”)
              (diff set (select set))
   (else null))
```

# Analogy and number?

- In simple MCMC over a rich concept space some transitions are very unlikely.
  - E.g. N-knower => CP-knower.
- Proposals via Syntactic analogy + recursion detection?

A CP-knower lexicon:

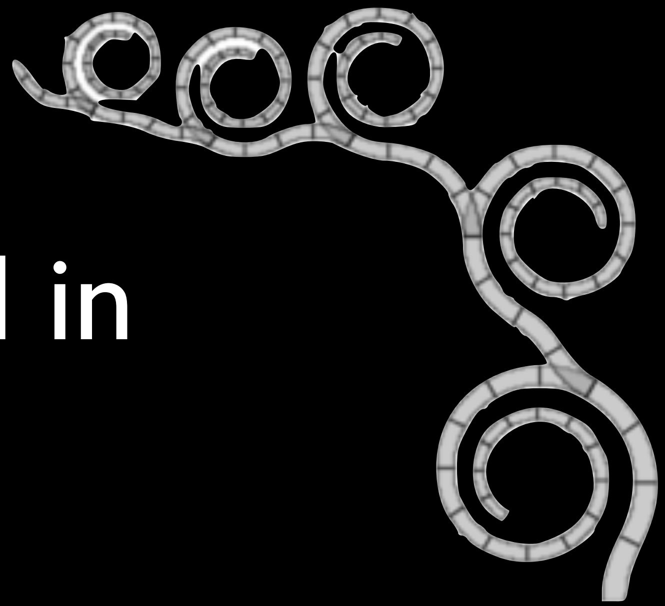
```
(define lexicon
 (λ (word)
  (if (= word "one")
      (λ (set)
       (empty? (diff set (select set))))
      (λ (set)
        ((lexicon (prev word))
              (diff set (select set))))))))
  ("three" (λ (set)
            ((lexicon "two")
             (diff set (select set)))
  (else null))
```

# Outline

If concepts are probabilistic programs,
then concept learning is *probabilistic program induction*.

- Boolean categories
- Quantified concepts
- Natural number concepts
- Generative kinds
- Program induction

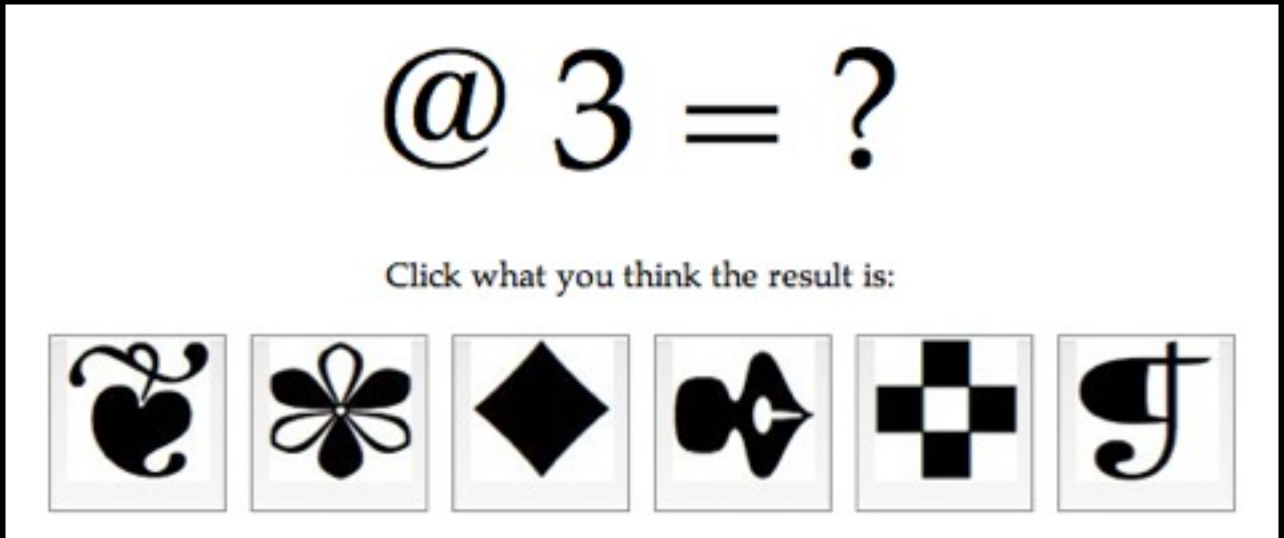
# An evolving *conceptome*



- Concepts (and theories) are represented in the PLoT, induced in response to evidence.
  - Learned concepts become ‘effective primitives’ for new concepts.

# Role of existing concepts

- 8 concepts all have same "logical" structure.
- In Number domain people have pre-existing concepts that are useful to varying degrees.

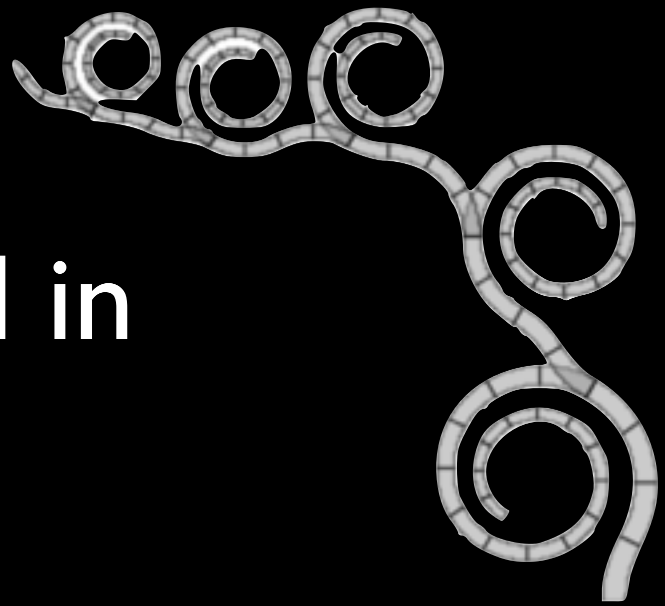



- Learning can be enhanced or retarded by existing concepts.

| | $@(x)$ when $x$ is: | | | | | | $@(x)$ when $x$ is: | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Label | 1 | 2 | 3 | 4 | 5 | 6 | 1 | 2 | 3 | 4 | 5 | 6 |
| $+7$ | 8 | 9 | 10 | 11 | 12 | 13 | ✾ | ⧓ | ❦ | ✚ | ℑ | ◆ |
| $14-$ | 13 | 12 | 11 | 10 | 9 | 8 | ◆ | ℑ | ✚ | ❦ | ⧓ | ✾ |
| $+8*$ | 9 | 10 | 11 | 12 | 13 | 8 | ⧓ | ❦ | ✚ | ℑ | ◆ | ✾ |
| *mem* | 12 | 9 | 13 | 10 | 11 | 8 | ℑ | ⧓ | ◆ | ❦ | ✚ | ✾ |

Ouyang & Goodman (in prep)

# An evolving *conceptome*



- Concepts (and theories) are represented in the PLoT, induced in response to evidence.
  - Learned concepts become 'effective primitives' for new concepts.
  - The evidence comes from perception, language, and social interaction.
  - Rich trajectories of conceptual change emerge.

# Conclusion

Probabilistic inference

Generative models

Compositional representations

Probabilistic language of thought